ADVANCED AND EVER ADVANCING MITSUBISHI ELECTRIC

MITSUBISHI

PROGRAMMABLE CONTROLLERS

MELSEC-F

PROGRAMMING MANUAL Ⅱ

THE FX SERIES OF PROGRAMMABLE CONTROLLER
(FX_{1S}, FX_{1N}, FX_{2N}, FX_{2NC})

FX

FX Series Programmable Controllers

Programming Manual

Manual number : JY992D88101
Manual revision : A
Date : April 2000

Foreword

- This manual contains text, diagrams and explanations which will guide the reader in the correct programming and operation of the PLC.
- Before attempting to install or use the PLC this manual should be read and understood.
- If in doubt at any stage of the installation of the PLC always consult a professional electrical engineer who is qualified and trained to the local and national standards which apply to the installation site.
- If in doubt about the operation or use of the PLC please consult the nearest Mitsubishi Electric distributor.
- This manual is subject to change without notice.

FAX BACK - Combined Programming Manual (J)

Mitsubishi has a world wide reputation for its efforts in continually developing and pushing back the frontiers of industrial automation. What is sometimes overlooked by the user is the care and attention to detail that is taken with the documentation. However,to continue this process of improvement, the comments of the Mitsubishi users are always welcomed. This page has been designed for you,the reader,to fill in your comments and fax them back to us. We look forward to hearing from you.

Please tick the box of your choice;

Fax numbers:		Your name.....
Mitsubishi Electric....		.....
America	(01) 847-478-2253	Your company .....
Australia	(02) 638-7072	.....
Germany	(0 21 02) 4 86-1 12	Your location: .....
South Africa	(0111) 444-8304	.....
United Kingdom	(01707) 278-695	

What condition did the manual arrive in?	☐Good	☐Minor damage	☐Unusable
Will you be using a folder to store the manual?		☐Yes	☐No
What do you think to the manual presentation?		☐Tidy	☐Un-friendly
Are the explanations understandable?	☐Yes	☐Not too bad	☐Unusable

Which explanation was most difficult to understand: .....
.....

Are there any diagrams which are not clear? ☐Yes ☐No

If so,which:.....

What do you think to the manual layout? ☐Good ☐Not too bad ☐Un-helpful

If there one thing you would like to see improved,what is it?.....
.....
.....

Could you find the information you required easily using the index and/or the contents,if possible please identify your experience: .....
.....
.....
.....
.....

Do you have any comments in general about the Mitsubishi manuals?.....
.....
.....
.....
.....

Thank you for taking the time to fill out this questionnaire. We hope you found both the product and this manual easy to use.

Guidelines for the Safety of the User and Protection of the Programmable Controller (PLC)

This manual provides information for the use of the FX family of PLC's. The manual has been written to be used by trained and competent personnel. The definition of such a person or persons is as follows;

a) Any engineer who is responsible for the planning, design and construction of automatic equipment using the product associated with this manual should be of a competent nature, trained and qualified to the local and national standards required to fulfill that role. These engineers should be fully aware of all aspects of safety with regards to automated equipment.

b) Any commissioning or service engineer must be of a competent nature, trained and qualified to the local and national standards required to fulfill that job. These engineers should also be trained in the use and maintenance of the completed product. This includes being completely familiar with all associated documentation for the said product. All maintenance should be carried out in accordance with established safety practices.

c) All operators of the completed equipment should be trained to use that product in a safe and coordinated manner in compliance to established safety practices. The operators should also be familiar with documentation which is connected with the actual operation of the completed equipment.

Note : the term 'completed equipment' refers to a third party constructed device which contains or uses the product associated with this manual.

Note's on the Symbols used in this Manual

At various times through out this manual certain symbols will be used to highlight points of information which are intended to ensure the users personal safety and protect the integrity of equipment. Whenever any of the following symbols are encountered its associated note must be read and understood. Each of the symbols used will now be listed with a brief description of its meaning.

Hardware Warnings

1) Indicates that the identified danger **WILL** cause physical and property damage.

2) Indicates that the identified danger could **POSSIBLY** cause physical and property damage.

3) Indicates a point of further interest or further explanation.

Software Warnings

4) Indicates special care must be taken when using this element of software.

5) Indicates a special point which the user of the associate software element should be aware of.

6) Indicates a point of interest or further explanation.

Contents

Chapter Contents

FX1S	FX1N	FX2N	FX2NC

1. Introduction

1.1 Overview

1) Scope of this manual
This manual gives details on all aspects of operation and programming for FX1S, FX1N, FX2N and FX2NC programmable controllers (PLCs). For all information relating to the PLC hardware and installation, refer to the appropriate manual supplied with the unit.

2) How to use this manual
This manual covers all the functions of the highest specification Programmable (Logic) Controller (PLC). For this reason, the following indicator is included in relevant section titles to show which PLCs that section applies to;

FX1S	FX1N	FX2N	FX2NC

Shaded boxes indicate the applicable PLC type

- "FX1S)" - All FX1S PLCs
- "FX1N" - All FX1N PLCs
- "FX2N" - All FX2N PLCs
- "FX2NC" - All FX2NC PLCs

FX1S	FX1N	FX2N	FX2NC

If an indicator box is half shaded, as shown to the left, this means that not all the functions described in the current section apply to that PLC. The text explains in further detail or makes an independent reference.

If there are no indicator boxes then assume the section applies to all PLC types unless otherwise stated.

3) FX family
This is a generic term which is often used to describe all Programmable Controllers without identifying individual types or model names.

4) CPU version numbers and programming support
As Mitsubishi upgrades each model different versions have different capabilities.
   - Please refer to section 1.4 for details about peripheral support for each model.

1.2 What is a Programmable Controller?

FX1S | FX1N | FX2N | FX2NC

A Programmable Logic Controller (PLC or programmable controller) is a device that a user can program to perform a series or sequence of events. These events are triggered by stimuli (usually called inputs) received at the PLC or through delayed actions such as time delays or counted occur-rences. Once an event triggers, it actuates in the outside world by switching ON or OFF electronic control gear or the physical actuation of devices. A programmable controller will continually 'loop' through its internal 'user defined' program waiting for inputs and giving outputs at the programmed specific times.

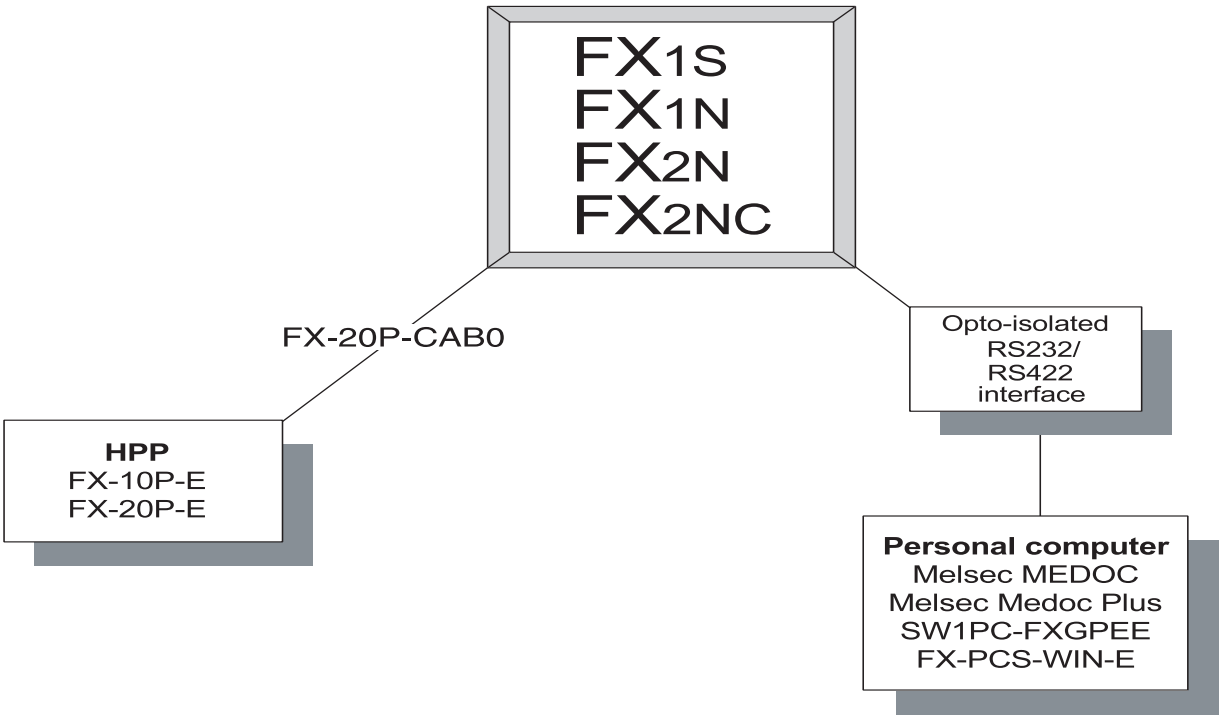
Note on terminology:

The term programmable controller is a generic word used to bring all the elements making the control system under one descriptive name. Sometimes engineers use the term 'Programmable Logic Controller', 'PLC' or 'programmable controller' to describe the same control system.
The construction of a programmable controller can be broken down into component parts. The element where the program is loaded, stored and processed is often known as the Main Processing Unit or MPU. Other terms commonly heard to describe this device are 'base unit', 'controller' and 'CPU'. The term CPU is a little misleading as todays more advanced products may contain local CPU devices. A Main CPU (or more correctly a Main Processing Unit) controls these local CPUs through a communication network or bus.

1.3 What do You Need to Program a PLC?

FX1S | FX1N | FX2N | FX2NC

A variety of tools are available to program the Mitsubishi FX family of PLCs. Each of these tools can use and access the instructions and devices listed in this manual for the identified PLC.



1.4 Special considerations for programming equipment

1.4.1 Current Generation CPU all versions

FX1S | FX1N | FX2N | FX2NC

The introduction of this CPU provides the FX user with many new devices and instructions. To use the full features of the current range of FX units the user must upgrade older software and hardware programming tools.
However, because of the downward compatibility of the current range, it is not necessary to upgrade existing programming tools up to the equivalent functionality of last generation FX CPU ver 3.30 units.

Peripherals Table		
Description	**Model Number**	**System software version with full support**
Hand held programmer (HHP)	FX-10P-E	from V 3.00
HHP cassette	FX-20P-MFXA-E	from V 3.00
Data access units	FX-10DU-E	from V 4.00
	FX-20DU-E	Supports up to FX devices only
	FX-25DU-E	from V 2.00
	FX-30DU-E	from V 3.00
	FX-40DU-E(S)	Supports up to FX devices only
	FX-40DU-TK-ES	from V 3.00
	FX-50DU-TK(S)-E	from V 2.10
	F930GOT-BWD	All versions
	F940GOT-SWD(LWD)-E	All versions

1.5 Assocciated Manuals

Manual name	Number
FX Base Unit Hardware	
FX1S Hardware manual	JY992D83901
FX1N Hardware manual	JY992D88201
FX2N Hardware manual	JY992D66301
FX2NC Hardware manual	JY992D76401
FX Programming	
FX0, FX0S, FX0N, FX, FX2C, FX2N, FX2NC Programming manual	JY992D48301
FX1S, FX1N, FX2N, FX2NC Programming manual Ⅱ	JY992D88101
FX Peripherals	
FX-10P-E Operation manual	JY992D33401
FX-20P-E Operation manual	JY992D19101
FX-10P, 20P-E Supplimentary manual	JY992D66901
FX-PCS-WIN-E Software manual	JY992D66501
FX Special Function Blocks	
FX0N-3A Users guide	JY992D49001
FX-4AD Users guide	JY992D52601
FX-2AD-PT Users guide	JY992D55701
FX-4AD-TC Users guide	JY992D55901
FX-2DA Users guide	JY992D52801
FX2N-2AD Users manual	JY992D74701
FX-4DA Users guide	JY992D61001
FX2N-4AD Users guide	JY992D65201
FX2N-4AD-TC Users guide	JY992D65501
FX2N-4AD-PT Users guide	JY992D65601
FX2N-4DA Users guide	JY992D65901
FX2N-2DA Users manual	JY992D74901
FX2N-2LC Users guide	JY992D85601
FX2N-2LC Users manual	JY992D85801
FX-485PC-IF Hardware manual	JY992D81801
FX/FX0N-485ADP Users guide	JY992D53201
FX-232ADP Users guide	JY992D48801
FX0N-232ADP Users guide	JY992D51301
FX2N-232BD Users guide	JY992D66001
FX2N-422BD Users guide	JY992D66101
FX2N-485BD Hardware manual	JY992D73401
FX2N-232IF Hardware manual	JY992D73501
FX Communication Users manual	JY992D69901
FX2N-CCL Users manual	JY992D71801
FX2N-16LNK-M Users manual	JY992D73701
FX0N-32NT-DP Users manual	JY992D61401
FX2N-32DP-IF Hardware manual	JY992D77101
FX2N-32DP-IF Users manual	JY992D79401
FX2N-32ASI-M Users manual	JY992D76901

Manual name	Number
FX DU, GOT and DM units	
FX-5DM Users manual	JY992D84901
FX-10DM Users manual	JY992D86401
FX Positioning	
FX-1HC Users guide	JY992D53001
FX2N/FX-1PG-E Users manual	JY992D65301
E-20P-E Operation manual	JY992D44901
FX2N-1HC Users guide	JY992D65401
FX2N-1RM-E-SET Users manual	JY992D71101
FX2N-10GM Users guide	JY992D77701
FX2N-20GM Users guide	JY992D77601
FX2N-10/20GM Hardware/Programming manual	JY992D77801
FX-PCS-VPS/WIN-E Software manual	JY992D86801

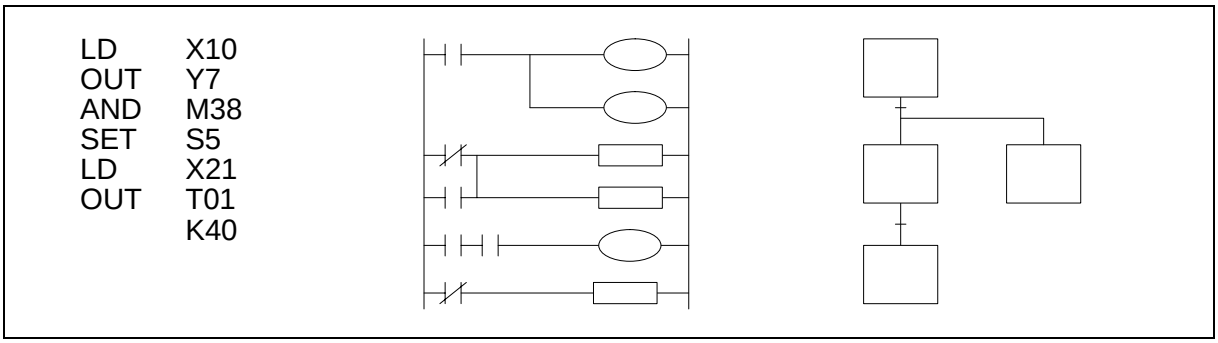
Memo

Chapter Contents

2. Basic Program Instructions

2.1 What is a Program?

A program is a connected series of instructions written in a language that the PLC can understand. There are three forms of program format; instruction, ladder and SFC/STL. Not all programming tools can work in all programming forms. Generally hand held programming panels only work with instruction format while most graphic programming tools will work with both instruction and ladder format. Specialist programming software will also allow SFC style programming.



Instruction format Ladder Format SFC Format

2.2 Outline of Basic Devices Used in Programming

There are six basic programming devices. Each device has its own unique use. To enable quick and easy identification each device is assigned a single reference letter;

- X: This is used to identify all direct, physical inputs to the PLC.
- Y: This is used to identify all direct, physical outputs from the PLC.
- T: This is used to identify a timing device which is contained within the PLC.
- C: This is used to identify a counting device which is contained within the PLC.
- M and S: These are used as internal operation flags within the PLC.

All of the devices mentioned above are known as ‘bit devices’. This is a descriptive title telling the user that these devices only have two states; ON or OFF, 1 or 0.

Detailed device information:

- Chapter 4 contains this information in detail. However, the above is all that is required for the rest of this chapter.

2.3 How to Read Ladder Logic

Ladder logic is very closely associated to basic relay logic. There are both contacts and coils that can be loaded and driven in different configurations. However, the basic principle remains the same.

A coil drives direct outputs of the PLC (ex. a Y device) or drives internal timers, counters or flags (ex. T, C, M and S devices). Each coil has associated contacts. These contacts are available in both “normally open” (NO) and “normally closed” (NC) configurations.

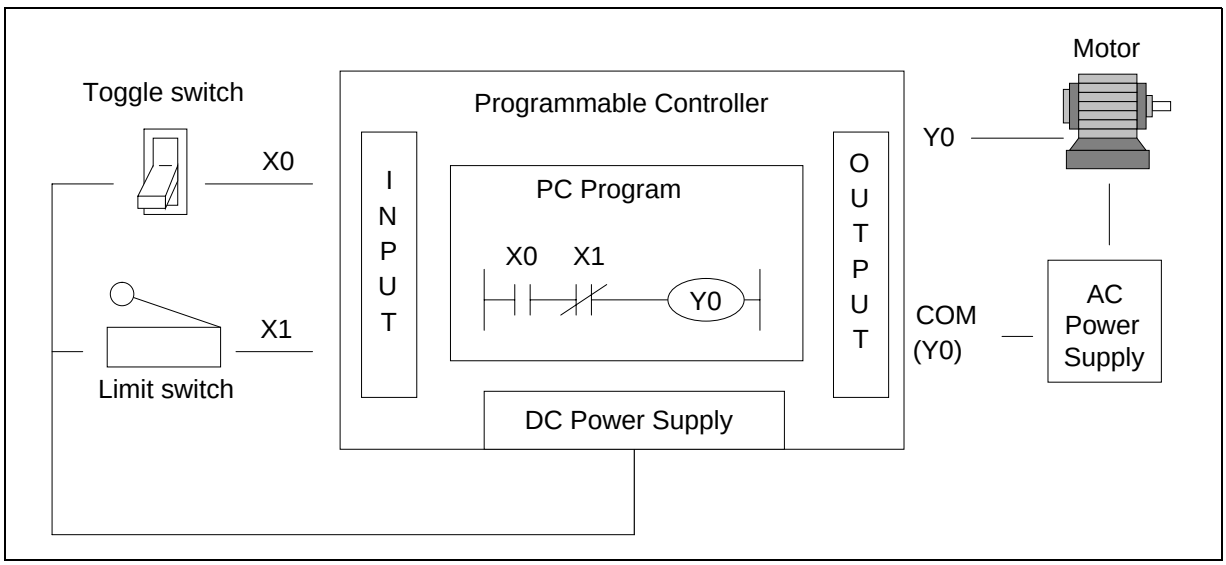
The term “normal(ly)” refers to the status of the contacts when the coil is not energized. Using a relay analogy, when the coil is OFF, a NO contact would have no current flow, that is, a load being supplied through a NO contact would not operate. However, a NC contact would allow current to flow, hence the connected load would be active.

Activating the coil reverses the contact status, that is, the current would flow in a NO contact and a NC contact would inhibit the flow.

Physical inputs to the PLC (X devices) have no programmable coil. These devices may only be used in a contact format (NO and NC types are available).

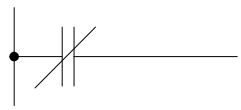
Example:

Because of the close relay association, ladder logic programs can be read as current flowing from the left vertical line to the right vertical line. This current must pass through a series of contact representations such as X0 and X1 in order to switch the output coil Y0 ON. Therefore, in the example shown, switching X0 ON causes the output Y0 to also switch ON. If however, the limit switch X1 is activates, the output Y0 turns OFF. This is because the connection between the left and the right vertical lines breaks so there is no current flow.

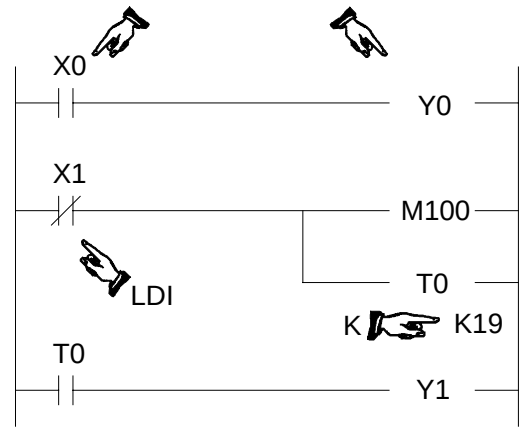


2.4 Load, Load Inverse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
LD (LoaD)	Initial logical operation contact type NO (normally open)		X, Y, M, S, T, C	1
LDI (LoaD Inverse)	Initial logical operation contact type NC (normally closed)		X, Y, M, S, T, C	1

Program example:




```
0   LD    X   0
1   OUT   Y   0
2   LDI   X   1
3   OUT   M   100
4   OUT   T   0
    SP    K   19
7   LD    T   0
8   OUT   Y   1
```

When using hand held programmers, the space key needs to be pressed to enable the constant to be entered.

Basic points to remember:

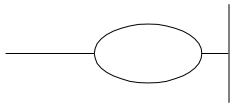
- Connect the LD and LDI instructions directly to the left hand bus bar.
- Or use LD and LDI instructions to define a new block of program when using the ORB and ANB instructions (see later sections).

The OUT instruction:

- For details of the OUT instruction (including basic timer and counter variations) please see over the following page.

2.5 Out

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
OUT (OUT)	Final logical operation type coil drive		Y, M, S, T, C	Y, M:1 S, special M coils: 2 T:3 C (16 bit): 3 C (32 bit): 5

Basic points to remember:

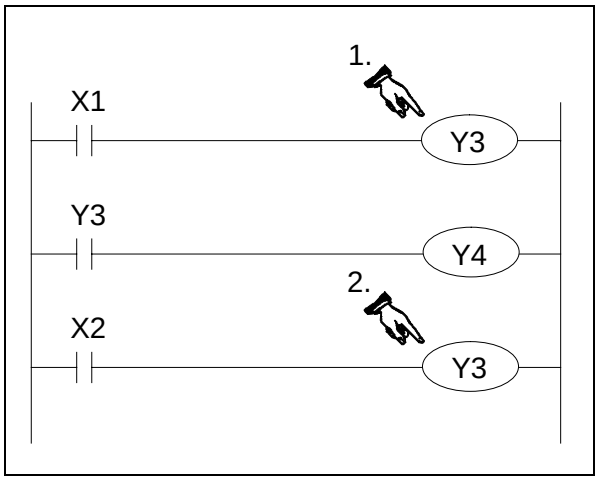
- Connect the OUT instruction directly to the right hand bus bar.
- It is not possible to use the OUT instruction to drive 'X' type input devices.
- It is possible to connect multiple OUT instructions in parallel (for example see the previous page; M100/T0 configuration)

2.5.1 Timer and Counter Variations

When configuring the OUT instruction for use as either a timer (T) or counter (C) a constant must also be entered. The constant is identified by the letter "K" (for example see previous page; T0 K19).
In the case of a timer, the constant "K" holds the duration data for the timer to operate, i.e. if a 100 msec timer has a constant of "K100" it will be (1005 100 msec) 10 seconds before the timer coil activates.
With counters, the constant identifies how many times the counter must be pulsed or triggered before the counter coil activates. For example, a counter with a constant of "8" must be triggered 8 times before the counter coil finally energizes.
The following table identifies some basic parameter data for various timers and counters;

Timer/Counter	Setting constant K	Actual setting	Program steps
1 msec Timer	1 to 32,767	0.001 to 32.767 sec	3
10 msec Timer		0.01 to 327.67 sec	
100 msec Timer		0.1 to 3276.7 sec	
16 bit Counter	1 to 32,767	1 to 32,767	
32 bit Counter	-2,147,483,648 to 2,147,483,647	-2,147,483,648 to 2,147,483,647	5

2.5.2 Double Coil Designation



Double or dual coiling is not a recommended practice. Using multiple output coils of the same device can cause the program operation to become unreliable. The example program shown opposite identifies a double coil situation; there are two Y3 outputs. The following sequence of events will occur when inputs X1 = ON and X2 = OFF;

1.The first Y3 tuns ON because X1 is ON. The contacts associated with Y3 also energize when the coil of output Y3 energizes. Hence, output Y4 turns ON.

2.The last and most important line in this program looks at the status of input X2.

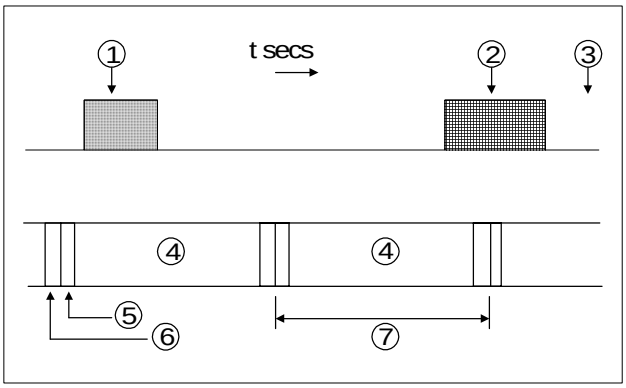
If this is NOT ON then the second Y3 coil does NOT activate. Therefore the status of the Y3 coil updates to reflect this new situation, i.e. it turns OFF. The final outputs are then Y3 = OFF and Y4 = ON.

Use of dual coils:

- Always check programs for incidents of dual coiling. If there are dual coils the program will not operate as expected - possibly resulting in physical damage.

The last coil effect:

- In a dual coil designation, the coil operation designated last is the effective coil. That is, it is the status of the previous coil that dictates the behavior at the current point in the program.



Input durations:

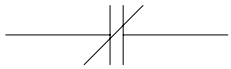
The ON or OFF duration of the PLC inputs must be longer than the operation cycle time of the PLC.
Taking a 10 msec (standard input filter) response delay into account, the ON/OFF duration must be longer than 20 msec if the operation cycle (scan time) is 10 msec.

Therefore, in this example, input pulses of more than 25Hz (1sec/(20msec ON + 20msec OFF)) cannot be sensed.
There are applied instructions provided to handle such high speed input requests.

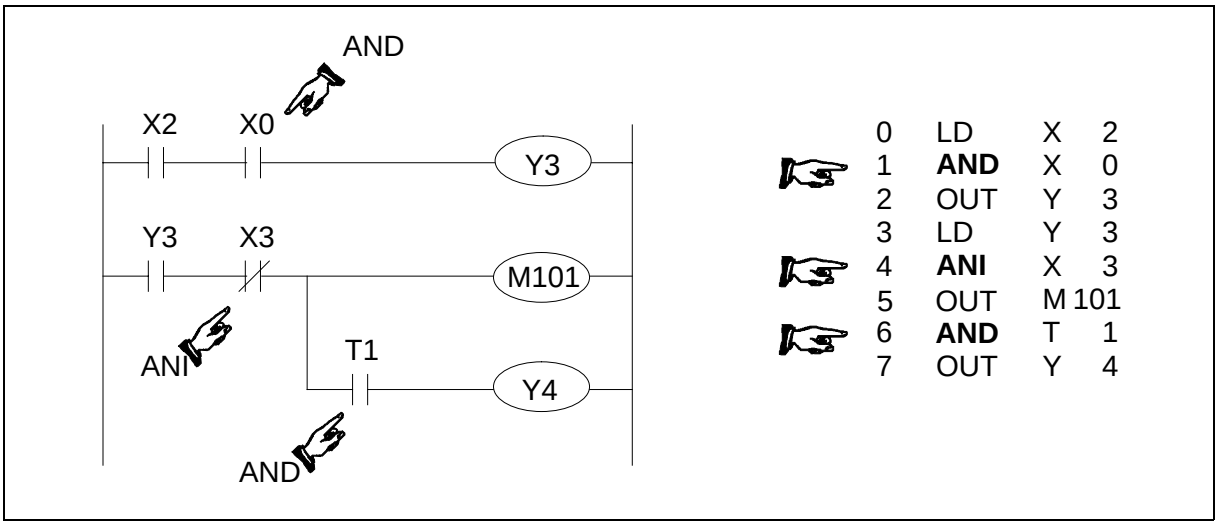
①: Input ON state NOT recognized
②: Input ON state recognized
③: Input OFF state NOT recognized
④: 1 program processing
⑤: Input processing
⑥: Output processing
⑦: A full program scan/operation cycle

2.6 And, And Inverse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
AND (AND)	Serial connection of NO (normally open) contacts		X, Y, M, S, T, C	1
ANI (AND Inverse)	Serial connection of NC (normally closed) contacts		X, Y, M, S, T, C	1

Program example:



Basic points to remember:

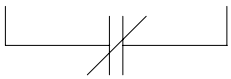
- Use the AND and ANI instructions for serial connection of contacts. As many contacts as required can be connected in series (see following point headed “Peripheral limitations”).
- The output processing to a coil, through a contact, after writing the initial OUT instruction is called a “follow-on” output (for an example see the program above; OUT Y4). Follow-on outputs are permitted repeatedly as long as the output order is correct.

Peripheral limitations:

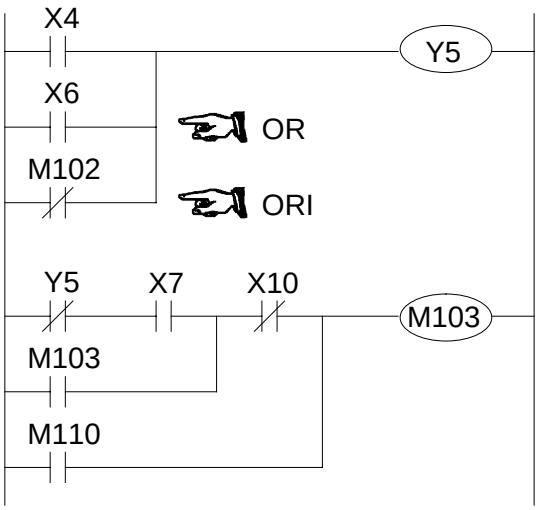
- The PLC has no limit to the number of contacts connected in series or in parallel. However, some programming panels, screens and printers will not be able to display or print the program if it exceeds the limit of the hardware. It is preferable for each line or rung of ladder program to contain up to a maximum of 10 contacts and 1 coil. Also, keep the number of follow-on outputs to a maximum of 24.

2.7 Or, Or Inverse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
OR (OR)	Parallel connection of NO (normally open) contacts		X, Y, M, S, T, C	1
ORI (OR Inverse)	Parallel connection of NC (normally closed) contacts		X, Y, M, S, T, C	1

Program example:




```
0   LD    X   4
1   OR    X   6
2   ORI   M   102
3   OUT   Y   5
4   LDI   Y   5
5   AND   X   7
6   OR    M   103
7   ANI   X   10
8   OR    M   110
9   OUT   M   103
```

Basic points to remember:

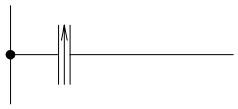
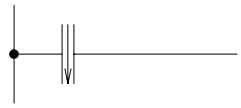
- Use the OR and ORI instructions for parallel connection of contacts. To connect a block that contains more than one contact connected in series to another circuit block in parallel, use an ORB instruction.
- Connect one side of the OR/ORI instruction to the left hand bus bar.

Peripheral limitations:

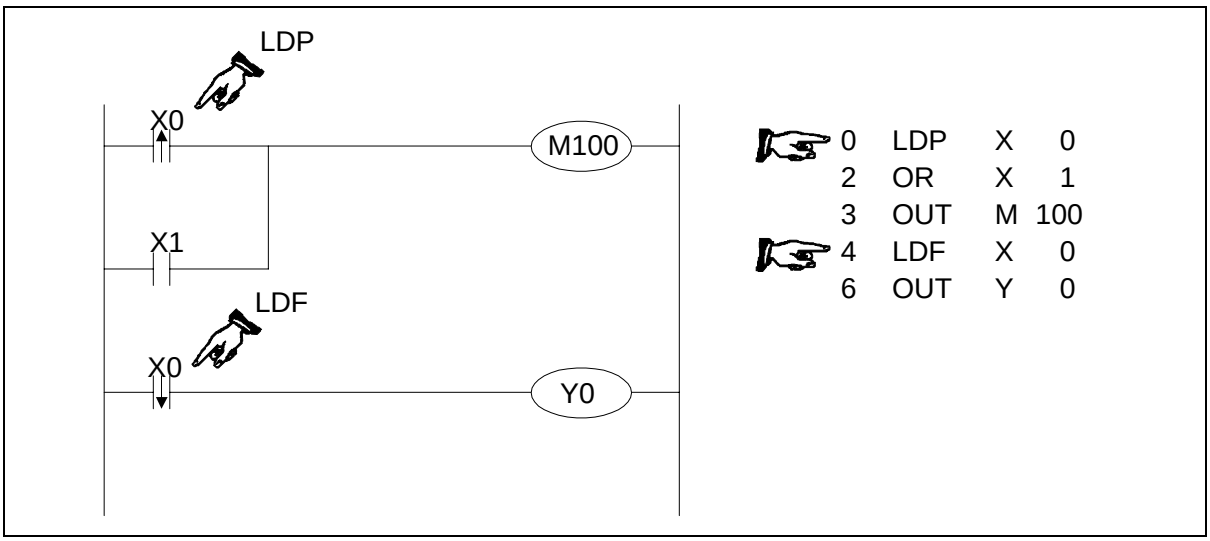
- The PLC has no limit to the number of contacts connected in series or in parallel. However, some programming panels, screens and printers will not be able to display or print the program if it exceeds the limit of the hardware. It is preferable for each line or rung of ladder program to contain up to a maximum of 10 contacts and 1 coil. Also keep number of follow-on outputs to a maximum of 24.

2.8 Load Pulse, Load Trailing Pulse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
LDP *(LoaDPulse)*	Initial logical operation - Rising edge pulse		X, Y, M, S, T, C	2
LDF *(LoaD Falling pulse)*	Initial logical operation Falling / trailing edge pulse		X, Y, M, S, T, C	2

Program example:



Basic points to remember:

- Connect the LDP and LDF instructions directly to the left hand bus bar.
- Or use LDP and LDF instructions to define a new block of program when using the ORB and ANB instructions (see later sections).
- LDP is active for one program scan after the associated device switches from OFF to ON.
- LDF is active for one program scan after the associated device switches from ON to OFF.

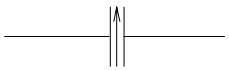
Single Operation flags M2800 to M3071:

- The pulse operation instructions, when used with auxiliary relays M2800 to M3071, only activate the first instruction encountered in the program scan, after the point in the program where the device changes. Any other pulse operation instructions will remain inactive.
- This is useful for use in STL programs (see chapter 3) to perform single step operation using a single device.
- Any other instructions (LD, AND, OR, etc.) will operate as expected.

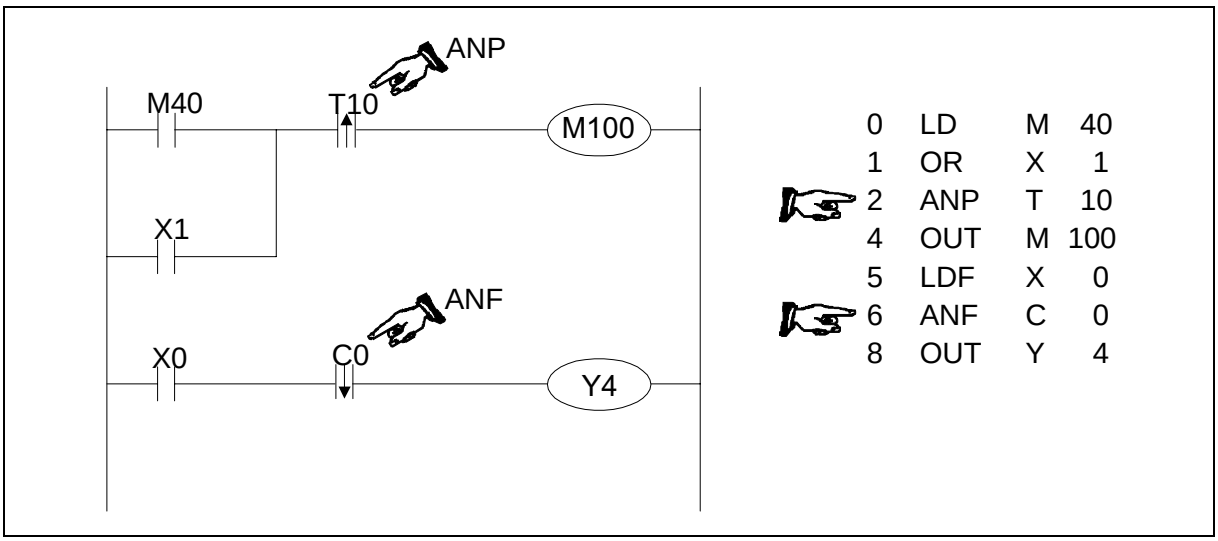
For more details please see page 4-5.

2.9 And Pulse, And Trailing Pulse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
ANP *(ANd Pulse)*	Serial connection of Rising edge pulse		X, Y, M, S, T, C	2
ANF *(ANd Falling pulse)*	Serial connection of Falling / trailing edge pulse		X, Y, M, S, T, C	2

Program example:



Basic points to remember:

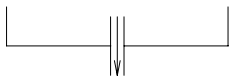
- Use the ANDP and ANDF instructions for the serial connection of pulse contacts.
- Usage is the same as for AND and ANI; see earlier.
- ANP is active for one program scan after the associated device switches from OFF to ON.
- ANF is active for one program scan after the associated device switches from ON to OFF.

Single operation flags M2800 to M3071:

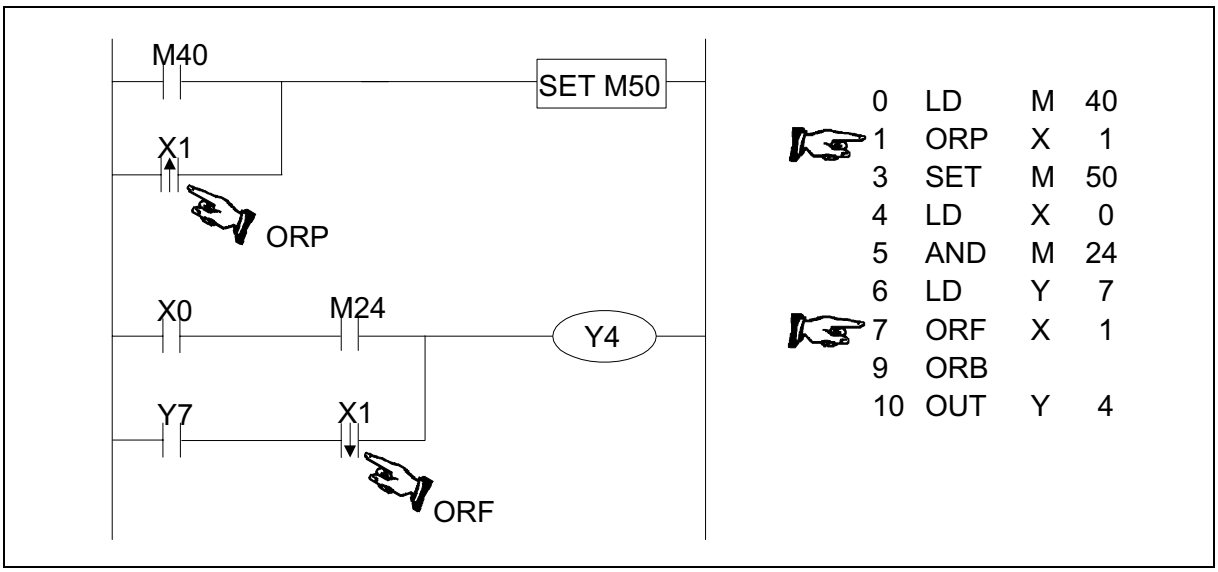
- When used with flags M2800 to M3071 only the first instruction will activate. For details see page 2-8

2.10 Or Pulse, Or Trailing Pulse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
ORP *(OR Pulse)*	Parallel connection of Rising edge pulse		X, Y, M, S, T, C	2
ORF *(OR Falling pulse)*	Parallel connection of Falling / trailing edge pulse		X, Y, M, S, T, C	2

Program example:



Basic points to remember:

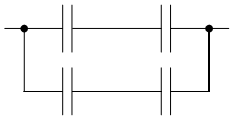
- Use the ORP and ORF instructions for the parallel connection of pulse contacts.
- Usage is the same as for OR and ORI; see earlier.
- ORP is active for one program scan after the associated device switches from OFF to ON.
- ORF is active for one program scan after the associated device switches from ON to OFF.

Single operation flags M2800 to M3071:

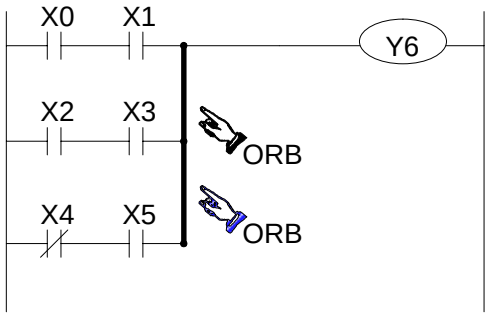
- When used with flags M2800 to M3071 only the first instruction will activate. For details see page 2-8

2.11 Or Block

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
ORB *(OR Block)*	Parallel connection of multiple contact circuits		N/A	1

Program example:



Recommended sequential programming method

```
0  LD   X 0
1  AND  X 1
2  LD   X 2
3  AND  X 3
4  ORB
5  LDI  X 4
6  AND  X 5
7  ORB
8  OUT  Y 6
```

Non-preferred batch programming method

```
0  LD   X 0
1  AND  X 1
2  LD   X 2
3  AND  X 3
4  LDI  X 4
5  AND  X 5
6  ORB
7  ORB
8  OUT  Y 6
```

Basic points to remember:

- An ORB instruction is an independent instruction and is not associated with any device number.
- Use the ORB instruction to connect multi-contact circuits (usually serial circuit blocks) to the preceding circuit in parallel. Serial circuit blocks are those in which more than one contact connects in series or the ANB instruction is used.
- To declare the starting point of the circuit block use a LD or LDI instruction. After completing the serial circuit block, connect it to the preceding block in parallel using the ORB instruction.

Batch processing limitations:

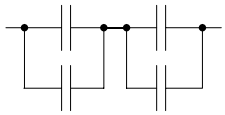
- When using ORB instructions in a batch, use no more than 8 LD and LDI instructions in the definition of the program blocks (to be connected in parallel). Ignoring this will result in a program error (see the right most program listing).

Sequential processing limitations:

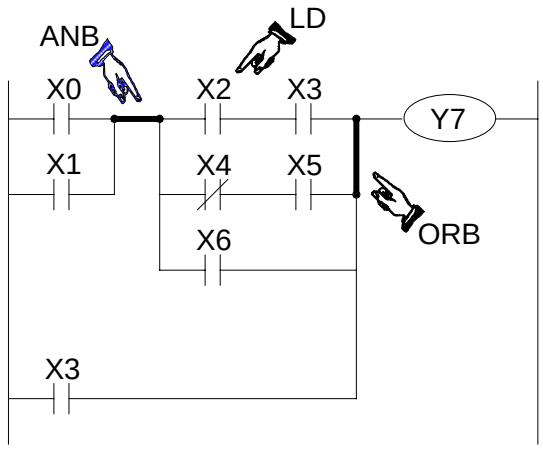
- There are no limitations to the number of parallel circuits when using an ORB instruction in the sequential processing configuration (see the left most program listing).

2.12 And Block

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
ANB *(ANd Block)*	Serial connection of multiple parallel circuits		N/A	1

Program example:



Recommended sequential programming method

	0	LD	X	0
	1	OR	X	1
	2	LD	X	2
	3	AND	X	3
	4	LDI	X	4
	5	AND	X	5
	6	ORB		
	7	OR	X	6
☞	8	**ANB**		
	9	OR	X	3
	10	OUT	Y	7

Basic points to remember:

- An ANB instruction is an independent instruction and is not associated with any device number
- Use the ANB instruction to connect multi-contact circuits (usually parallel circuit blocks) to the preceding circuit in series. Parallel circuit blocks are those in which more than one contact connects in parallel or the ORB instruction is used.
- To declare the starting point of the circuit block, use a LD or LDI instruction. After completing the parallel circuit block, connect it to the preceding block in series using the ANB instruction.

Batch processing limitations:

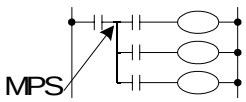
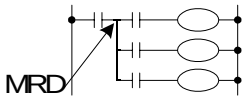
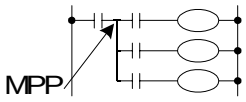
- When using ANB instructions in a batch, use no more than 8 LD and LDI instructions in the definition of the program blocks (to be connected in parallel). Ignoring this will result in a program error (see ORB explanation for example).

Sequential processing limitations:

- It is possible to use as many ANB instructions as necessary to connect a number of parallel circuit blocks to the preceding block in series (see the program listing).

2.13 MPS, MRD and MPP

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
MPS *(Point Store)*	Stores the current result of the internal PLC operations		N/A	1
MRD *(Read)*	Reads the current result of the internal PLC operations		N/A	1
MPP *(PoP)*	Pops (recalls and removes) the currently stored result		N/A	1

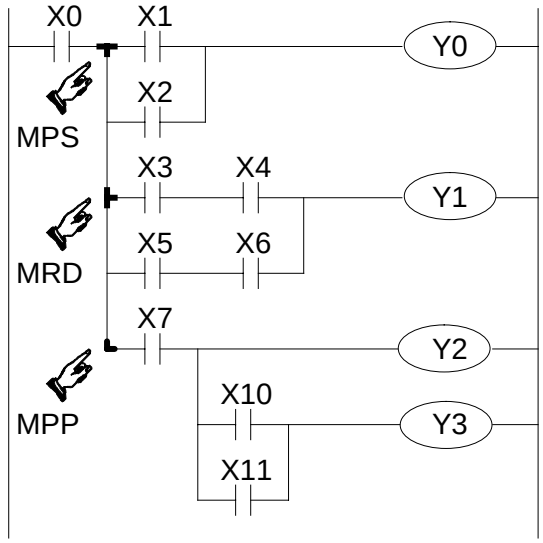
Basic points to remember:

- Use these instructions to connect output coils to the left hand side of a contact. Without these instructions connections can only be made to the right hand side of the last contact.
- MPS stores the connection point of the ladder circuit so that further coil branches can recall the value later.
- MRD recalls or reads the previously stored connection point data and forces the next contact to connect to it.
- MPP pops (recalls and removes) the stored connection point. First, it connects the next contact, then it removes the point from the temporary storage area.
- For every MPS instruction there MUST be a corresponding MPP instruction.
- The last contact or coil circuit must connect to an MPP instruction.
- At any programming step, the number of active MPS-MPP pairs must be no greater than 11.

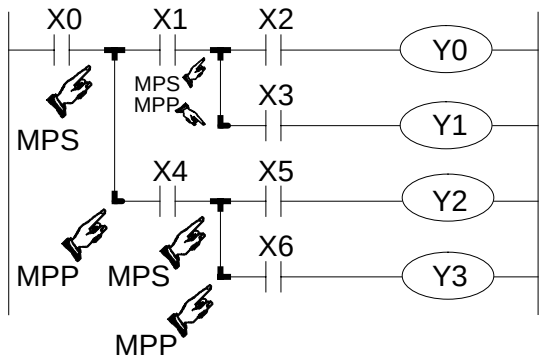
MPS, MRD and MPP usage:

- When writing a program in ladder format, programming tools automatically add all MPS, MRD and MPP instructions at the program conversion stage. If the generated instruction program is viewed, the MPS, MRD and MPP instructions are present.
- When writing a program in instruction format, it is entirely down to the user to enter all relevant MPS, MRD and MPP instructions as required.

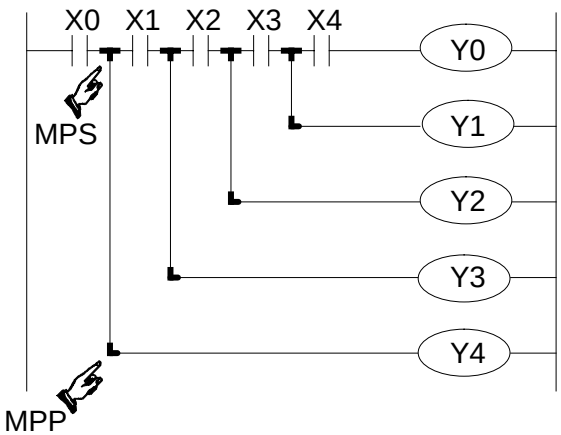
Multiple program examples:




```
 0  LD   X  0
 1  MPS
 2  LD   X  1
 3  OR   X  2
 4  ANB
 5  OUT  Y  0
 6  MRD
 7  LD   X  3
 8  AND  X  4
 9  LD   X  5
10  AND  X  6
11  ORB
12  ANB
13  OUT  Y  1
14  MPP
15  AND  X  7
16  OUT  Y  2
17  LD   X 10
18  OR   X 11
19  ANB
20  OUT  Y  3
```




```
 0  LD   X  0
 1  MPS
 2  AND  X  1
 3  MPS
 4  AND  X  2
 5  OUT  Y  0
 6  MPP
 7  AND  X  3
 8  OUT  Y  1
 9  MPP
10  AND  X  4
11  MPS
12  AND  X  5
13  OUT  Y  2
14  MPP
15  AND  X  6
16  OUT  Y  3
```



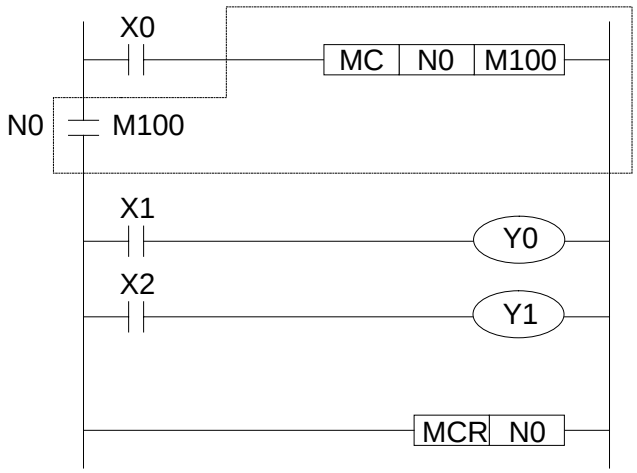

```
 0  LD   X  0
 1  MPS
 2  AND  X  1
 3  MPS
 4  AND  X  2
 5  MPS
 6  AND  X  3
 7  MPS
 8  AND  X  4
 9  OUT  Y  0
10  MPP
11  OUT  Y  1
12  MPP
13  OUT  Y  2
14  MPP
15  OUT  Y  3
16  MPP
17  OUT  Y  4
```

2.14 Master Control and Reset

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
MC *(Master Control)*	Denotes the start of a master control block		Y, M (no special M coils allowed) N denotes the nest level (N0 to N7)	3
MCR *(Master Control Reset)*	Denotes the end of a master control block		N denotes the nest level (N0 to N7) to be reset.	2

Program example:

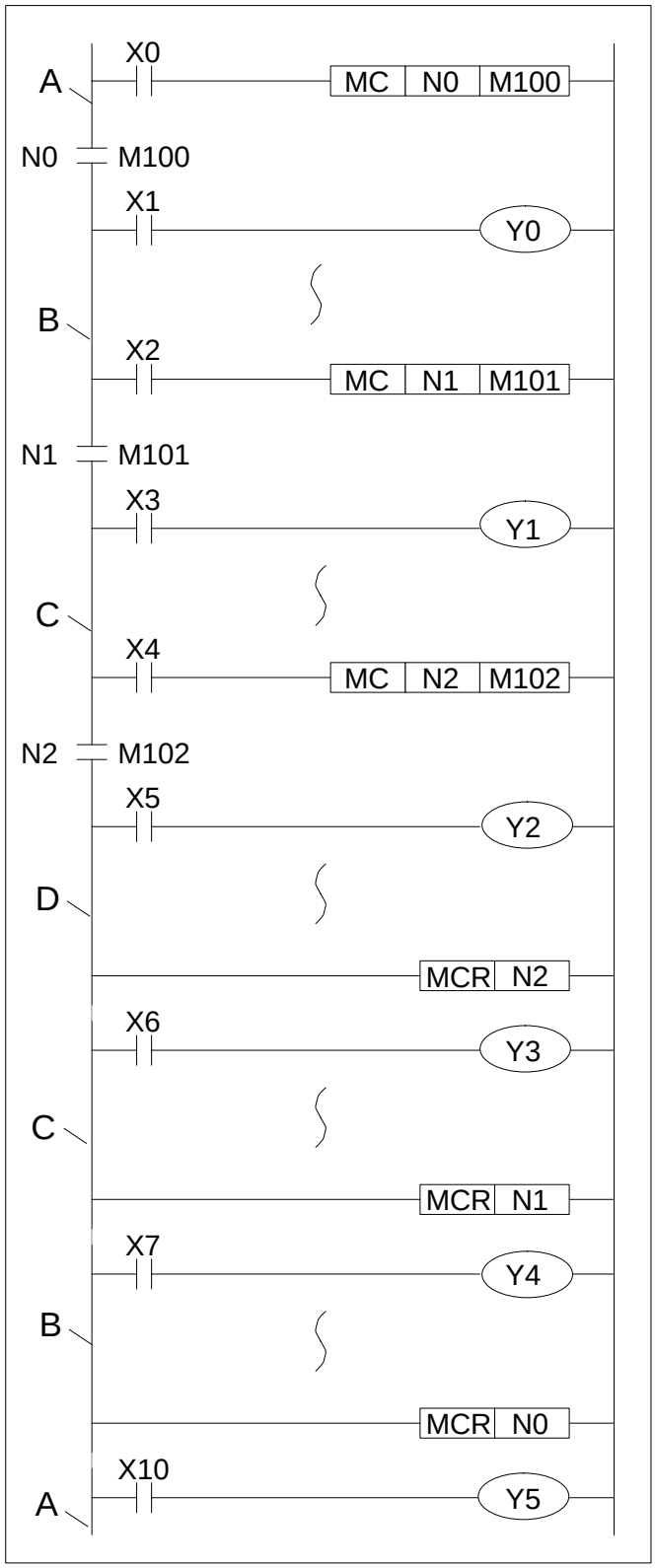



```
0   LD    X   0
1   MC    N   0
      SP      M 100
4   LD    X   1
5   OUT   Y   0
6   LD    X   2
7   OUT   Y   1
8   MCR   N   0
```

Note: SP - space key
N - nest level of MC (N0 to N7)

Basic points to remember:

- After the execution of an MC instruction, the bus line (LD, LDI point) shifts to a point after the MC instruction. An MCR instruction returns this to the original bus line.
- The MC instruction also includes a nest level pointer N. Nest levels are from the range N0 to N7 (8 points). The top nest level is ‘0’ and the deepest is ‘7’.
- The MCR instruction resets each nest level. When a nest level is reset, it also resets ALL deeper nest levels. For example, MCR N5 resets nest levels 5 to 7.
- When input X0=ON, all instructions between the MC and the MCR instruction execute.
- When input X0=OFF, none of the instruction between the MC and MCR instruction execute; this resets all devices except for retentive timers, counters and devices driven by SET/RST instructions.
- The MC instruction can be used as many times as necessary, by changing the device number Y and M. Using the same device number twice is processed as a double coil (see section 2.5.2). Nest levels can be duplicated but when the nest level resets, ALL occurrences of that level reset and not just the one specified in the local MC.



Nested MC program example:

Level N0: Bus line (B) active when X0 is ON.

Level N1: Bus line (C) active when both X0 and X2 are ON.

Level N2: Bus line (D) active when X0,X2 and X4 are ON.

Level N1: MCRN2 executes and restores bus line (C). If the MCR had reset N0 then the original bus bar (A) would now be active as all master controls below nest level 0 would reset.

Level N0: MCRN1 executes and restores bus line (B).

Initial state: MCR N0 executes and restores the initial bus line (A).

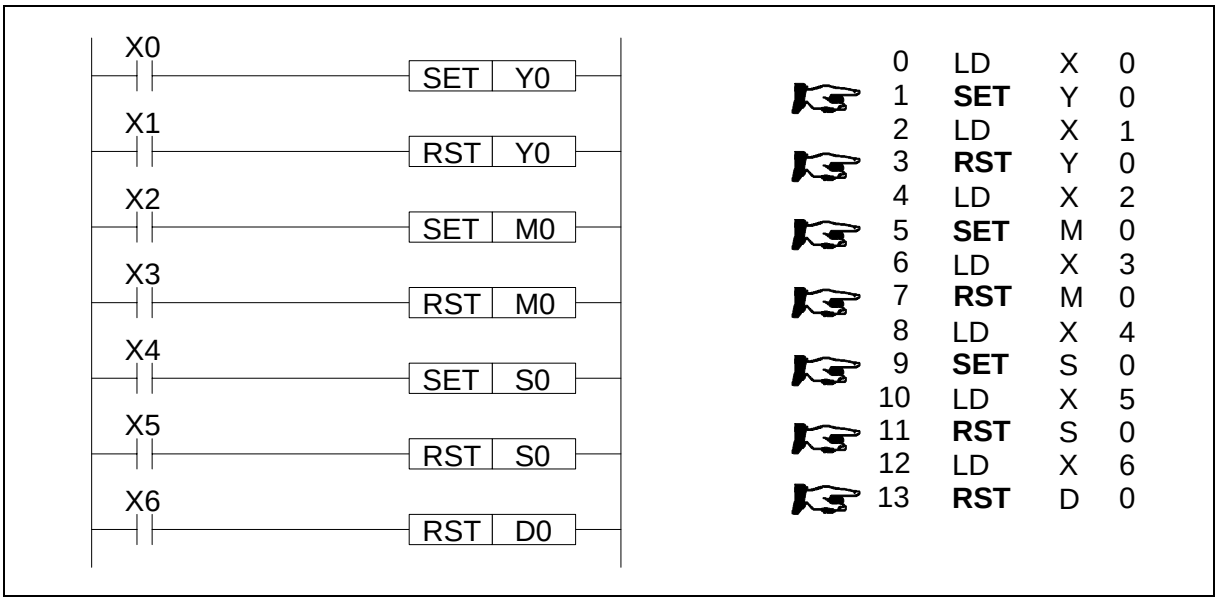
Output Y5 turns ON/OFF according to the ON/OFF state of X10, regardless of the ON/OFF status of inputs X0, X2 or X4.

2.15 Set and Reset

FX1S | FX1N | FX2N | FX2NC

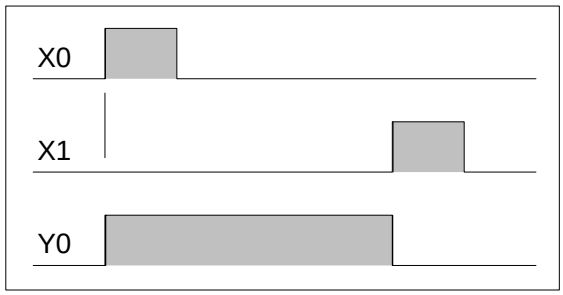
Mnemonic	Function	Format	Devices	Program steps
SET *(SET)*	Sets a bit device permanently ON		Y, M, S	Y,M:1 S, special M coils:2
RST *(ReSeT)*	Resets a bit device permanently OFF		Y, M, S, D, V, Z (see section 2.16 for timers and counters T,C)	D, special D registers, V and Z:3

Program example:



Basic points to remember:

- Turning ON X0 causes Y0 to turn ON. Y0 remains ON even after X0 turns OFF.
- Turning ON X1 causes Y0 to turn OFF. Y0 remains OFF even after X1 turns OFF.
- SET and RST instructions can be used for the same device as many times as necessary. However, the last instruction activated determines the current status.



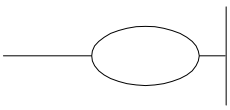
- It is also possible to use the RST instruction to reset the contents of data devices such as data registers, index registers etc. The effect is similar to moving 'K0' into the data device.

Resetting timers and counters:

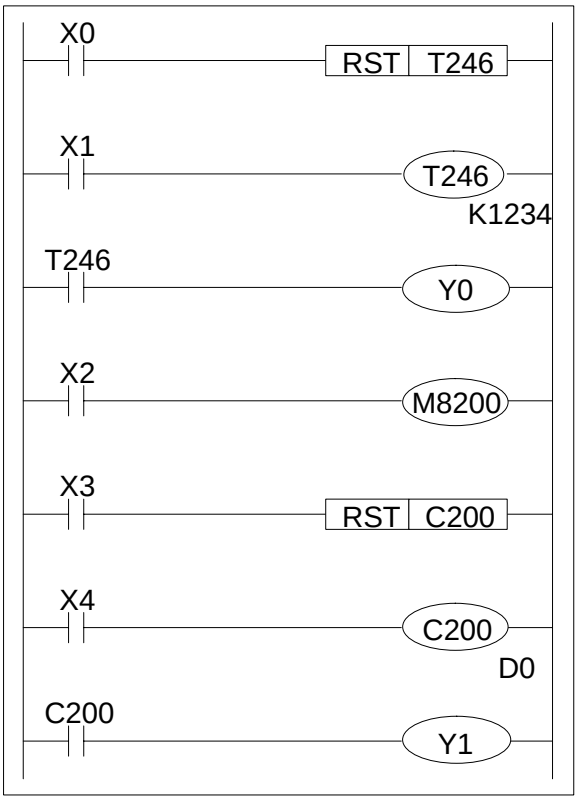
- Please see next page.

2.16 Timer, Counter (Out & Reset)

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
OUT *(OUT)*	Driving timer or counter coils		T, C	*32 bit counters:5 Others: 3*
RST *(ReSeT)*	Resets timer and counter, coils contacts and current values		T, C (see section 2.15 for other resetable devices)	*T, C:2*

Program example:



2.16.1Basic Timers, Retentive Timers And Counters

These devices can all be reset at any time by driving the RST instruction (with the number of the device to be reset).
On resetting, all active contacts, coils and current value registers are reset for the selected device. In the example, T246, a 1msec retentive timer, is activate while X1 is ON. When the current value of T246 reaches the preset 'K' value, i.e. 1234, the timer coil for T246 will be activated. This drives the NO contact ON. Hence, Y0 is switched ON. Turning ON X0 will reset timer T246 in the manner described previously.
Because the T246 contacts are reset, the output Y0 will be turned OFF.

Retentive timers:

- For more information on retentive timers please see page 4-17.

2.16.2 Normal 32 bit Counters

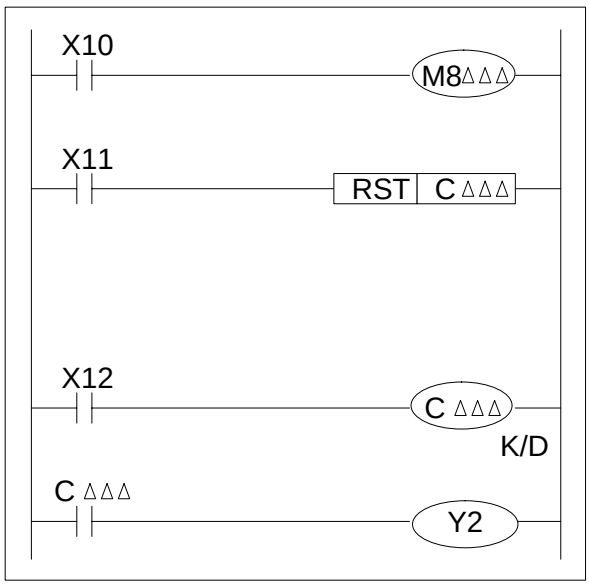
The 32 bit counter C200 counts (up-count, down-count) according to the ON/OFF state of M8200. In the example program shown on the previous page C200 is being used to count the number of OFF ~ ON cycles of input X4.
The output contact is set or reset depending on the direction of the count, upon reaching a value equal (in this example) to the contents of data registers D1,D0 (32 bit setting data is required for a 32 bit counter).
The output contact is reset and the current value of the counter is reset to '0' when input X3 is turned ON.

32 bit counters:

- For more information on 32 bit counters please see page 4-21.

2.16.3 High Speed Counters

High speed counters have selectable count directions. The directions are selected by driving the appropriate special auxiliary M coil. The example shown to the right works in the following manner; when X10 is ON, counting down takes place. When X10 is OFF counting up takes place.
In the example the output contacts of counter CΔΔΔ and its associated current count values are reset to "0" when X11 is turned ON. When X12 is turned ON the driven counter is enabled. This means it will be able to start counting its assigned input signal (this will not be X12 - high speed counters are assigned special input signals, please see page 4-22)



.

Availability of devices:

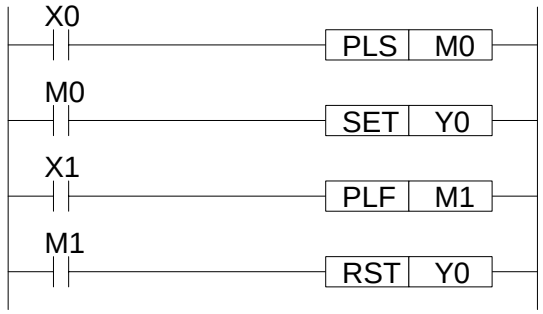
- Not all devices identified here are available on all programmable controllers. Ranges of active devices may vary from PLC to PLC. Please check the specific availability of these devices on the selected PLC before use. For more information on high speed counters please see page 4-22. For PLC device ranges please see chapter 8.

2.17 Leading and Trailing Pulse

FX1S | FX1N | FX2N | FX2NC

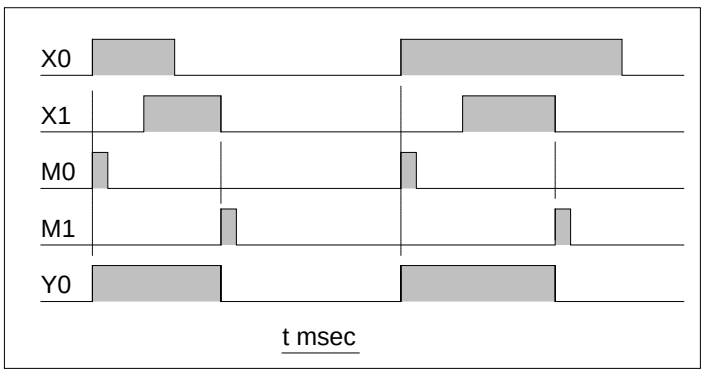
Mnemonic	Function	Format	Devices	Program steps
PLS *(PuLSe)*	Rising edge pulse		Y, M (no special M coils allowed)	2
PLF *(PuLse Falling)*	Falling / trailing edge pulse		Y, M (no special M coils allowed)	2

Program example:




```
0   LD    X   0
1   PLS   M   0
3   LD    M   0
4   SET   Y   0
5   LD    X   1
6   PLF   M   1
8   LD    M   1
9   RST   Y   0
```

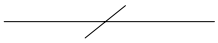
Basic points to remember:



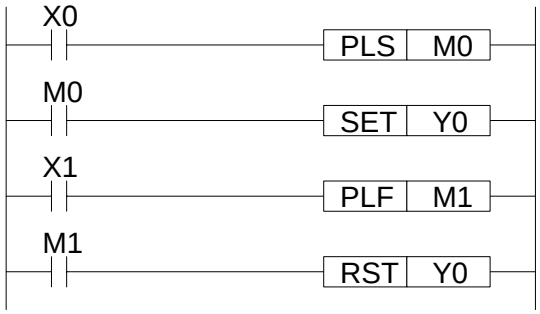
- When a PLS instruction is executed, object devices Y and M operate for one operation cycle after the drive input signal has turned ON.
- When a PLF instruction is executed, object devices Y and M operate for one operation cycle after the drive input signal has turned OFF.
- When the PLC status is changed from RUN to STOP and back to RUN with the input signals still ON, PLS M0 is operated again. However, if an M coil which is battery backed (latched) was used instead of M0 it would not re-activate. For the battery backed device to be re-pulsed, its driving input (ex. X0) must be switched OFF during the RUN/STOP/RUN sequence before it will be pulsed once more.

2.18 Inverse

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
INV (Inverse)	Invert the current result of the internal PLC operations		N/A	1

Program example:




```
0  LD   X 0
1  PLS  M 0
3  LD   M 0
4  SET  Y 0
5  LD   X 1
6  PLF  M 1
8  LD   M 1
9  RST  Y 0
```

Basic points to remember:

- The INV instruction is used to change (invert) the logical state of the current ladder network at the inserted position.
- Usage is the same as for AND and ANI; see earlier.

Usages for INV

- Use the invert instruction to quickly change the logic of a complex circuit.
  It is also useful as an inverse operation for the pulse contact instructions LDP, LDF, ANP, etc.

2.19 No Operation

FX1S | FX1N | FX2N | FX2NC

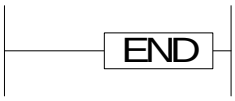
Mnemonic	Function	Format	Devices	Program steps
NOP *(No Operation)*	No operation or null step	N/A	N/A	1

Basic points to remember:

- Writing NOP instructions in the middle of a program minimizes step number changes when changing or editing a program.
- It is possible to change the operation of a circuit by replacing programmed instructions with NOP instructions.
- Changing a LD, LDI, ANB or an ORB instruction with a NOP instruction will change the circuit considerably; quite possibly resulting in an error being generated.
- After the program 'all clear operation' is executed, all of the instructions currently in the program are over written with NOP's.

2.20 End

FX1S | FX1N | FX2N | FX2NC

Mnemonic	Function	Format	Devices	Program steps
END *(END)*	Forces the current program scan to end		N/A	1

Basic points to remember:

- Placing an END instruction in a program forces that program to end the current scan and carry out the updating processes for both inputs and outputs.
- Inserting END instructions in the middle of the program helps program debugging as the section after the END instruction is disabled and isolated from the area that is being checked. Remember to delete the END instructions from the blocks which have already been checked.
- When the END instruction is processed the PLCs watchdog timer is automatically refreshed.

A program scan:

- A program scan is a single processing of the loaded program from start to finish, This includes updating all inputs, outputs and watchdog timers. The time period for one such process to occur is called the scan time. This will be dependent upon program length and complexity. Immediately the current scan is completed the next scan begins. The whole process is a continuous cycle. Updating of inputs takes place at the beginning of each scan while all outputs are updated at the end of the scan.

MEMO

Chapter Contents

3. STL Programming

FX1S | FX1N | FX2N | FX2NC

This chapter differs from the rest of the contents in this manual as it has been written with a training aspect in mind. STL/SFC programming, although having been available for many years, is still misunderstood and misrepresented. We at Mitsubishi would like to take this opportunity to try to correct this oversight as we see STL/SFC programming becoming as important as ladder style programming.

3.1 What is STL, SFC And IEC1131 Part 3?

The following explanation is very brief but is designed to quickly outline the differences and similarities between STL, SFC and IEC1131 part 3.
In recent years Sequential Function Chart (or SFC) style programming (including other similar styles such as Grafcet and Funktionplan) have become very popular through out Europe and have prompted the creation of IEC1131 part 3.
The IEC1131 SFC standard has been designed to become an interchangeable programming language. The idea being that a program written to IEC1131 SFC standards on one manufacturers PLC can be easily transferred (converted) for use on a second manufacturers PLC.
STL programming is one of the basic programming instructions included in all FX PLC family members. The abbreviation STL actually means STep Ladder programming.
STL programming is a very simple concept to understand yet can provide the user with one of the most powerful programming techniques possible. The key to STL lies in its ability to allow the programmer to create an operational program which 'flows' and works in almost exactly the same manner as SFC. This is not a coincidence as this programming technique has been developed deliberately to achieve an easy to program and monitor system.
One of the key differences to Mitsubishi's STL programming system is that it can be entered into a PLC in 3 formats. These are:

I) Instruction - a word/mnemonic entry system

II) Ladder - a graphical program construction method using a relay logic symbols

III) SFC - a flow chart style of STL program entry (similar to SFC)

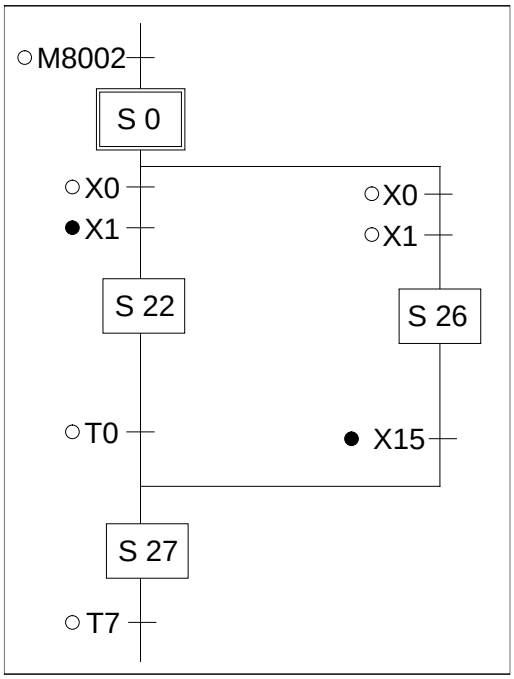
Examples of these programming methods can be seen on page 2-1.

General note:

- IEC1131-3: 03.1993 Programmable controllers; part 3: programming languages.
  The above standard is technically identical to the 'Euro-Norm'
  EN61131-3: 07.1993

3.2 How STL Operates

As previously mentioned, STL is a system which allows the user to write a program which functions in much the same way as a flow chart, this can be seen in the diagram opposite.
STL derives its strength by organizing a larger program into smaller more manageable parts. Each of these parts can be referred to as either a state or a step. To help identify the states, each is given a unique identification number. These numbers are taken from the state relay devices (see page 4-6 for more details).



3.2.1 Each step is a program

Each state is completely isolated from all other states within the whole program. A good way to envisage this, is that each state is a separate program and the user puts each of those programs together in the order that they require to perform their task. Immediately this means that states can be reused many times and in different orders. This saves on programming time AND cuts down on the number of programming errors encountered.

A Look Inside an STL

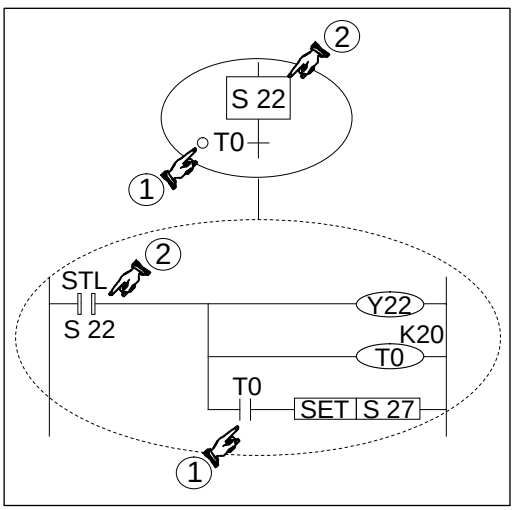
On initial inspection the STL program looks as if it is a rather basic flow diagram. But to find out what is really happening the STL state needs to be put 'under a microscope' so to speak. When a single state is examined in more detail, the sub-program can be viewed.

With the exception of the STL instruction, it will be immediately seen that the STL sub-program looks just like ordinary programming.

① The STL instruction is shown as a 'fat' normally open contact.
All programming after an STL instruction is only active when the associated state coil is active.

② The transition condition is also written using standard programming.

This idea re-enforces the concept that STL is really a method of sequencing a series of events or as mentioned earlier 'of joining lots of smaller programs together'.

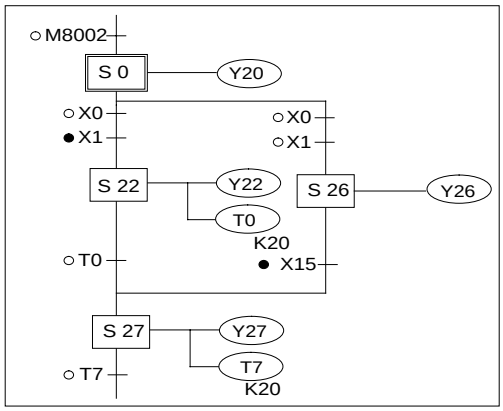


Combined SFC Ladder representation

Sometimes STL programs will be written in hard copy as a combination of both flow diagram and internal sub-program. (example shown below).
Identification of contact states

- Please note the following convention is used:
  - O Normally Open contact
  - ● Normally Closed contact

  Common alternatives are 'a' and 'b' identifiers for Normally Open, Normally Closed states or often a line drawn over the top of the Normally Closed contact name is used, e.g. X000.

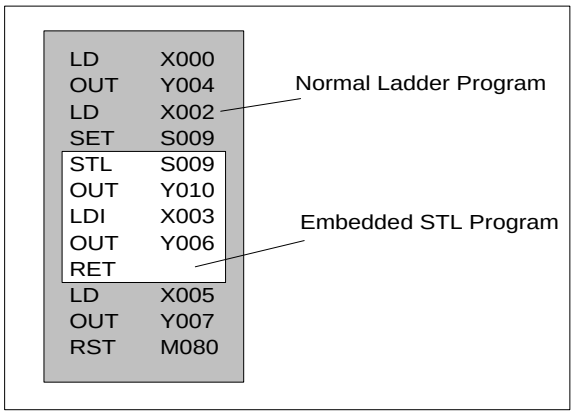


3.3 How To Start And End An STL Program

Before any complex programming can be undertaken the basics of how to start and more importantly how to finish an STL program need to be examined.

3.3.1 Embedded STL programs

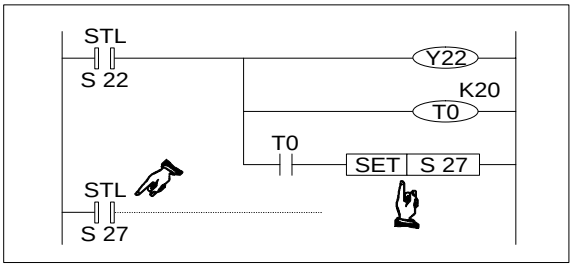
An STL style program does not have to entirely replace a standard ladder logic program. In fact it might be very difficult to do so. Instead small or even large section of STL program can be entered at any point in a program. Once the STL task has been completed the program must go back to processing standard program instructions until the next STL program block. Therefore, identifying the start and end of an STL program is very important.



3.3.2 Activating new states

Once an STL step has been selected, how is it used and how is the program 'driven'?
This is not so difficult, if it is considered that for an STL step to be active its associated state coil must be ON. Hence, to start an STL sequence all that has to be done is to drive the relevant state ON.

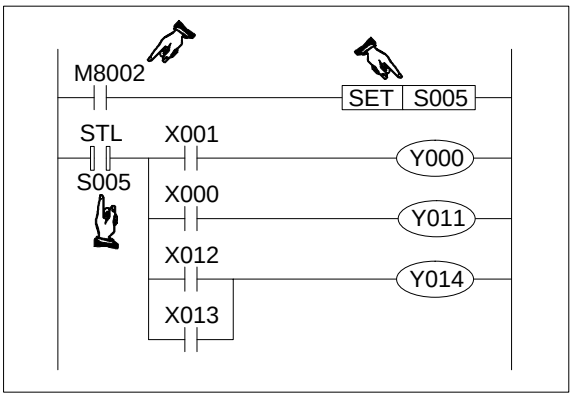
There are many different methods to drive a state, for example the initial state coils could be pulsed, SET or just included in an OUT instruction. However, within Mitsubishi's STL programming language an STL coil which is SET has a different meaning than one that is included in an OUT instruction.



Note: For normal STL operation it is recommended that the states are selected using the SET instruction. To activate an STL step its state coil is SET ON.

Initial Steps

For an STL program which is to be activated on the initial power up of the PLC, a trigger similar to that shown opposite could be used, i.e. using M8002 to drive the setting of the initial state.
The STL step started in this manner is often referred to as the initial step. Similarly, the step activated first for any STL sequence is also called the initial step.

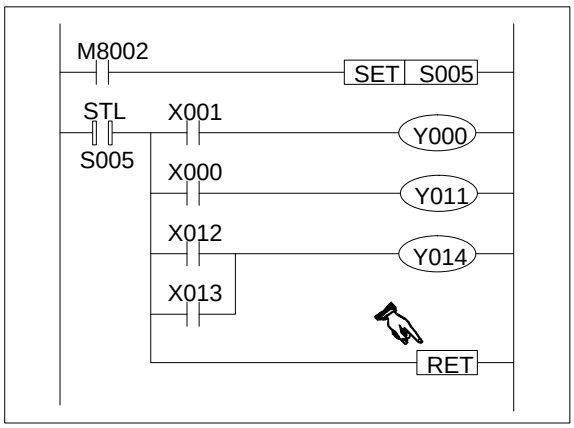


3.3.3 Terminating an STL Program

Once an STL program has been started the programmable controllers CPU will process all following instructions as being part of that STL program. This means that when a second program scan is started the normal instructions at the beginning of the program are considered to be within the STL program. This is obviously incorrect and the CPU will proceed to identify a programming error and disable the programmable controllers operation.
This scenario may seem a little strange but it does make sense when it is considered that the STL program must return control to the ladder program after STL operation is complete. This means the last step in an STL program needs to be identified in some way.

Returning to Standard Ladder

This is achieved by placing a RET or RETurn instruction as the last instruction in the last STL step of an STL program block.
This instruction then returns programming control to the ladder sequence.



Note: The RET instruction can be used to separate STL programs into sections, with standard ladder between each STL program. For display of STL in SFC style format the RET instruction is used to indicate the end of a complete STL program.

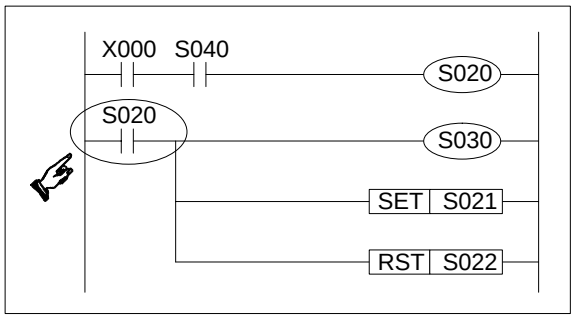
3.4 Moving Between STL Steps

To activate an STL step the user must first drive the state coil. Setting the coil has already been identified as a way to start an STL program, i.e. drive an initial state. It was also noted that using an OUT statement to driving a state coil has a different meaning to the SET instruction. These difference will now be explained:

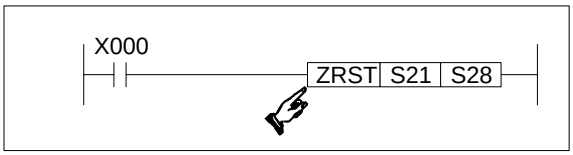
3.4.1 Using SET to drive an STL coil

- SET is used to drive an STL state coil to make the step active. Once the current STL step activates a second following step, the source STL coil is reset. Hence, although SET is used to activate a state the resetting is automatic.

  However, if an STL state is driven by a series of standard ladder logic instructions, i.e. not a preceding STL state, then standard programming rules apply.
  In the example shown opposite S20 is not reset even after S30 or S21 have been driven. In addition, if S20 is turned OFF, S30 will also stop operating. This is because S20 has not been used as an STL state. The first instruction involving the status of S20 is a standard LoaD instruction and NOT an STL instruction.



Note: If a user wishes to forcibly reset an STL step, using the RST or ZRST (FNC 40) instructions would perform this task.



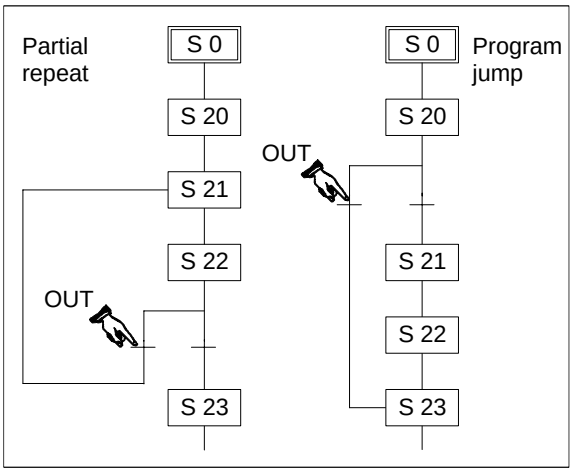
- SET is used to drive an immediately following STL step which typically will have a larger STL state number than the current step.
- SET is used to drive STL states which occur within the enclosed STL program flow, i.e. SET is not used to activate a state which appears in an unconnected, second STL flow diagram.

3.4.2 Using OUT to drive an STL coil

This has the same operational features as using SET. However, there is one major function which SET is not used. This is to make what is termed ‘distant jumps’.

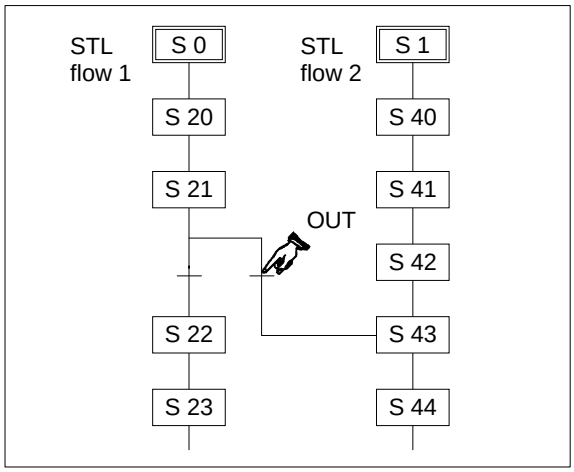
OUT is used for loops and jumps

If a user wishes to ‘jump’ back up a program, i.e. go back to a state which has already been processed, the OUT instruction would be used with the appropriate STL state number.
Alternatively the user may wish to make a large ‘jump’ forwards skipping a whole section of STL programmed states.



Out is used for distant jumps

If a step in one STL program flow was required to trigger a step in a second, separate STL program flow the OUT instruction would be used.



Note: Although it is possible to use SET for jumps and loops use of OUT is needed for display of STL in SFC like structured format.

3.5 Rules and Techniques For STL programs

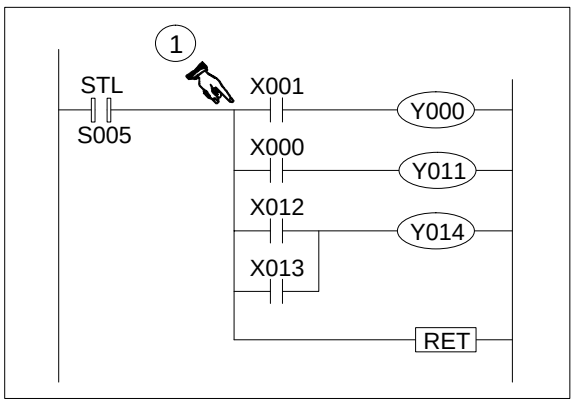
It can be seen that there are a lot of advantages to using STL style programming but there are a few points a user must be aware of when writing the STL sub-programs.
These are highlighted in this section.

3.5.1 Basic Notes On The Behavior Of STL programs

- When an STL state becomes active its program is processed until the next step is triggered. The contents of the program can contain all of the programming items and features of a standard ladder program, i.e. LoaD, AND OR, OUT, ReSeT etc., as well as applied instructions.

- When writing the sub-program of an STL state, the first vertical 'bus bar' after the STL instruction can be considered in a similar manner as the left hand bus bar of a standard ladder program.

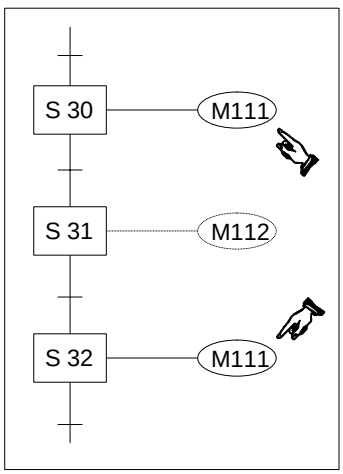
  Each STL step makes its own bus bar. This means that a user, cannot use an MPS instruction directly after the STL instruction (see ①), i.e. There needs to be at least a single contact before the MPS instruction.



Note: Using out coils and even applied instructions immediately after an STL instruction is permitted.

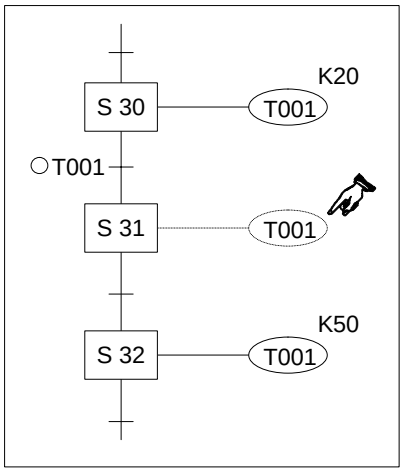
- In normal programming using dual coils is not an acceptable technique. However repetition of a coil in separate STL program blocks is allowed.

  This is because the user can take advantage of the STL's unique feature of isolating all STL steps except the active STL steps.
  This means in practice that there will be no conflict between dual coils. The example opposite shows M111 used twice in a single STL flow.



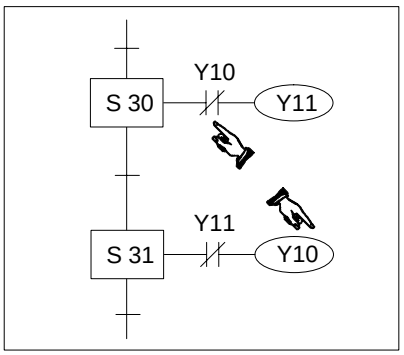
Caution: The same coil should NOT be programmed in steps that will be active at the same time as this will result in the same problem as other dual coils.

- When an STL step transfers control to the next STL step there is a period (one scan) while both steps are active. This can cause problems with dual coils; particularly timers.
  If timers are dual coiled care must be taken to ensure that the timer operation is completed during the active STL step.
  If the same timer is used in consecutive steps then it is possible that the timer coil is never deactivated and the contacts of the timer will not be reset leading to incorrect timer operation.
  The example opposite identifies an unacceptable use of timer T001. When control passes from S30 to S31 T001 is not reset because its coil is still ON in the new step.



Note: As a step towards ensuring the correct operation of the dual timers they should not be used in consecutive STL steps.
Following this simple rule will ensure each timer will be reset correctly before its next operation.

- As already mentioned, during the transfer between steps, the current and the selected steps will be simultaneously active for one program scan. This could be thought of as a hand over or handshaking period.
  This means that if a user has two outputs contained in consecutive steps which must NOT be active simultaneously they must be interlocked. A good example of this would be the drive signals to select a motors rotation direction. In the example Y11 and Y10 are shown interlocked with each other.



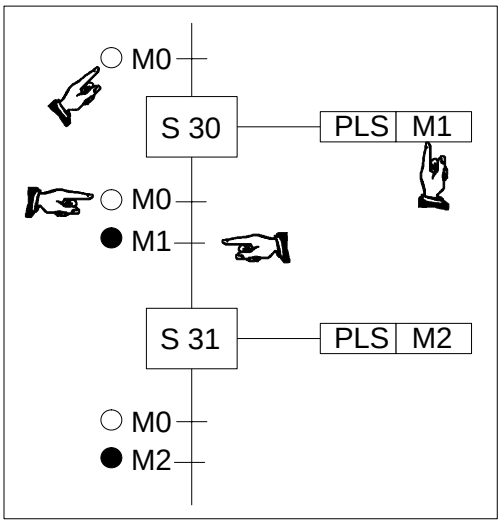
3.5.2 Single Signal Step Control

Transferring between active STL steps can be controlled by a single signal. There are two methods the user can program to achieve this result.

Method 1 - Using locking devices

FX1S | FX1N | FX2N | FX2NC

In this example it is necessary to program separate locking devices, and the controlling signal must only pulse ON. This is to prevent the STL programs from running through.
The example shown below identifies the general program required for this method.



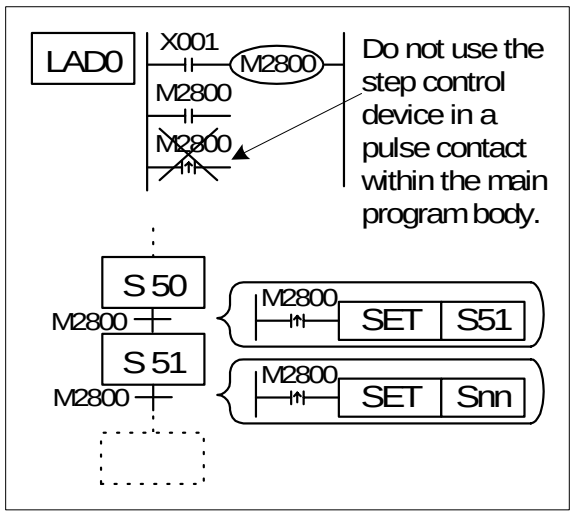
- S30 is activated when M0 is first pulsed ON.
- The operation of M1 prevents the sequence from continuing because although M0 is ON, the transfer requirements, need M0 to be ON and M1 to be OFF.
- After one scan the pulsed M0 and the 'lock' device M1 are reset.
- On the next pulse of M0 the STL step will transfer program control from S31 to the next step in a similar manner. This time using M2 as the 'lock' device because dual coils in successive steps is not allowed.
- The reason for the use of the 'lock' devices M1 and M2 is because of the handshaking period when both states involved in the transfer of program control are ON for 1 program scan. Without the 'locks' it would be possible to immediately skip through all of the STL states in one go!

Method 2 - Special Single Pulse Flags

FX1S | FX1N | FX2N | FX2NC

Using the pulse contacts (LDP, LDF, ANP, etc.) and a special range of M devices (M2800 to M3071) the same result as method 1 can be achieved. The special feature of these devices prevents run through of the states, as only the first occurrence of the LDP instruction will activate.

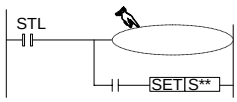
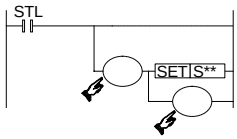
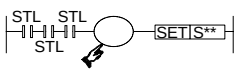
The example program below shows the necessary instructions.



- Assume S50 is already active.
- When X01 activates M2800, this in turn activates the LDP M2800 instruction in S50 and the flow moves on to step S51.
- The LDP M2800 instruction in the transition part of S51 does not execute because this is the second occurrence of M2800 in a pulse contact.
- When X01 next activates M2800, the LDP instruction in S51 is the first occurrence because S50 is now inactive. Thus, control passes to the next step in the same manner.

3.6 Restrictions Of Some Instructions When Used With STL

Although STL can operate with most basic and applied instructions there are a few exceptions. As a general rule STL and MC-MCR programming formats should not be combined. Other instruction restrictions are listed in the table below.

Operational State			Basic Instructions		
			LD, LDI, AND, ANI, OR,ORI, NOP, OUT, SET, RST, PLS,PLF	ANB, ORB, MPS,MRD, MPP	MC, MCR
Initial and general states			✔	✔	✗
Branch-ing and merging states	Output processing		✔	✔	✗
	Transfer processing		✔	✗	✗

Restrictions on using applied instructions

- Most applied instructions can be used within STL programs. Attention must be paid to the way STL isolates each non-active step. It is recommended that when applied instructions are used their operation is completed before the active STL step transfers to the next step.

  Other restrictions are as follows:

  - FOR - NEXT structures can not contain STL program blocks.
  - Subroutines and interrupts can not contain STL program blocks.
  - STL program blocks can not be written after an FEND instruction.
  - FOR - NEXT instructions are allowed within an STL program with a nesting of up to 4 levels.

For more details please see the operational compatibility listed in the two tables on pages 7-12,7-13.

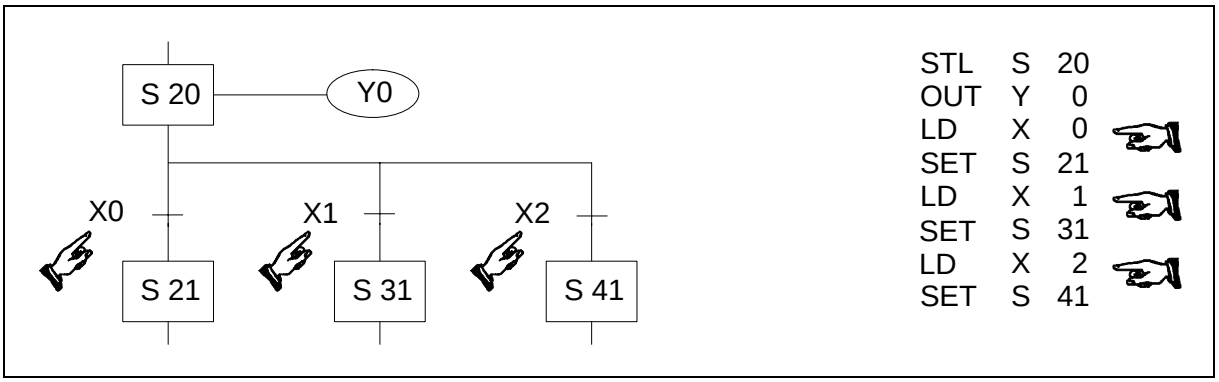
Using ‘jump’ operations with STL

- Although it is possible to use the program jump operations (CJ instruction) within STL program flows, this causes additional and often unnecessary program flow complications. To ensure easy maintenance and quick error finding it is recommended that users do not write jump instructions into their STL programs.

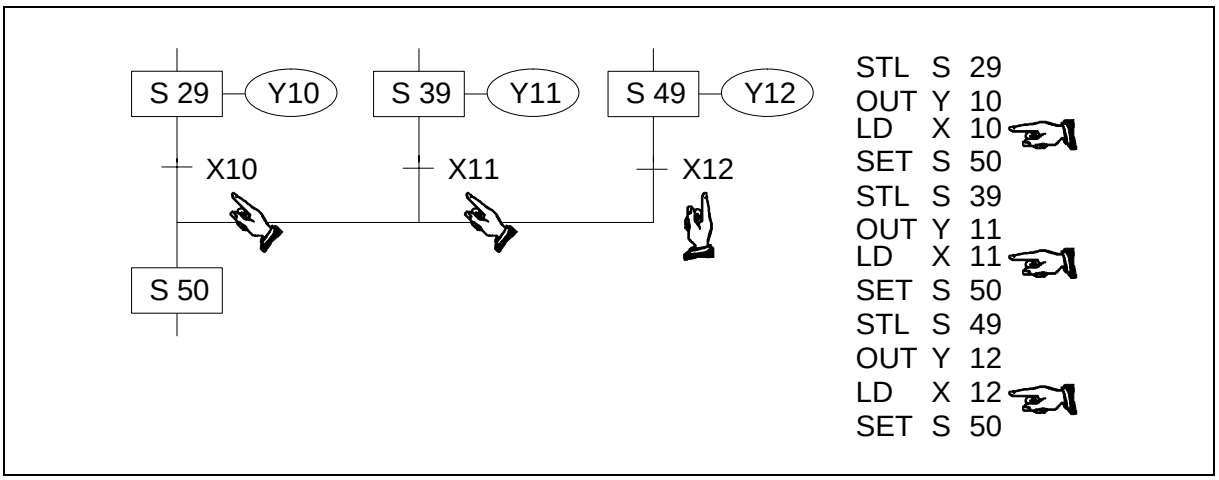
3.7 Using STL To Select The Most Appropriate Program

FX1S	FX1N	FX2N	FX2NC

So far STL has been considered as a simple flow charting programming language. One of STL's exceptional features is the ability to create programs which can have several operating modes. For example certain machines require a selection of 'manual' and 'automatic' modes, other machines may need the ability to select the operation or manufacturing processes required to produce products 'A', 'B', 'C', or 'D'. STL achieves this by allowing multiple program branches to originate from one STL state. Each branch is then programmed as an individual operating mode, and because each operating mode should act individually, i.e. there should be no other modes active; the selection of the program branch must be mutually exclusive. This type of program construction is called "Selective Branch Programming". An example instruction program can be seen below, (this is the sub-program for STL state S20 only) notice how each branch is SET by a different contact.



A programming construction to split the program flow between different branches is very useful but it would be more useful if it could be used with a method to rejoin a set of individual branches.



This type of STL program construction is called a "First State Merge" simply because the first state (in the example S29, S39 or S49) to complete its operation will cause the merging state (S50) to be activated. It should be noticed how each of the final STL states on the different program branches call the same "joining" STL state.

Limits on the number of branches

- Please see page 3-14 for general notes on programming STL branches.

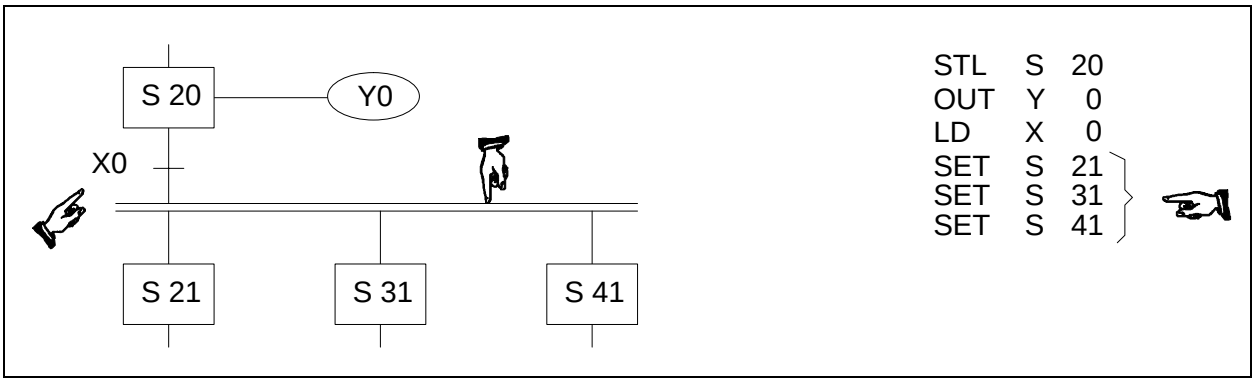
Notes on using the FX-PCS/AT-EE software

- Please see page 3-15 for precautions when using the FX-PCS-AT/EE software.

3.8 Using STL To Activate Multiple Flows Simultaneously

FX1S	FX1N	FX2N	FX2NC

In the previous branching technique, it was seen how a single flow could be selected from a group. The following methods describe how a group of individual flows can be activated simultaneously. Applications could include vending machines which have to perform several tasks at once, e.g. boiling water, adding different taste ingredients (coffee, tea, milk, sugar) etc. In the example below when state S20 is active and X0 is then switched ON, states S21, S31 and S41 are ALL SET ON as the next states. Hence, three separate, individual, branch flows are ‘set in motion’ from a single branch point. This programming technique is often called a ‘Parallel Branch’. To aid a quick visual distinction, parallel branches are marked with horizontal, parallel lines.

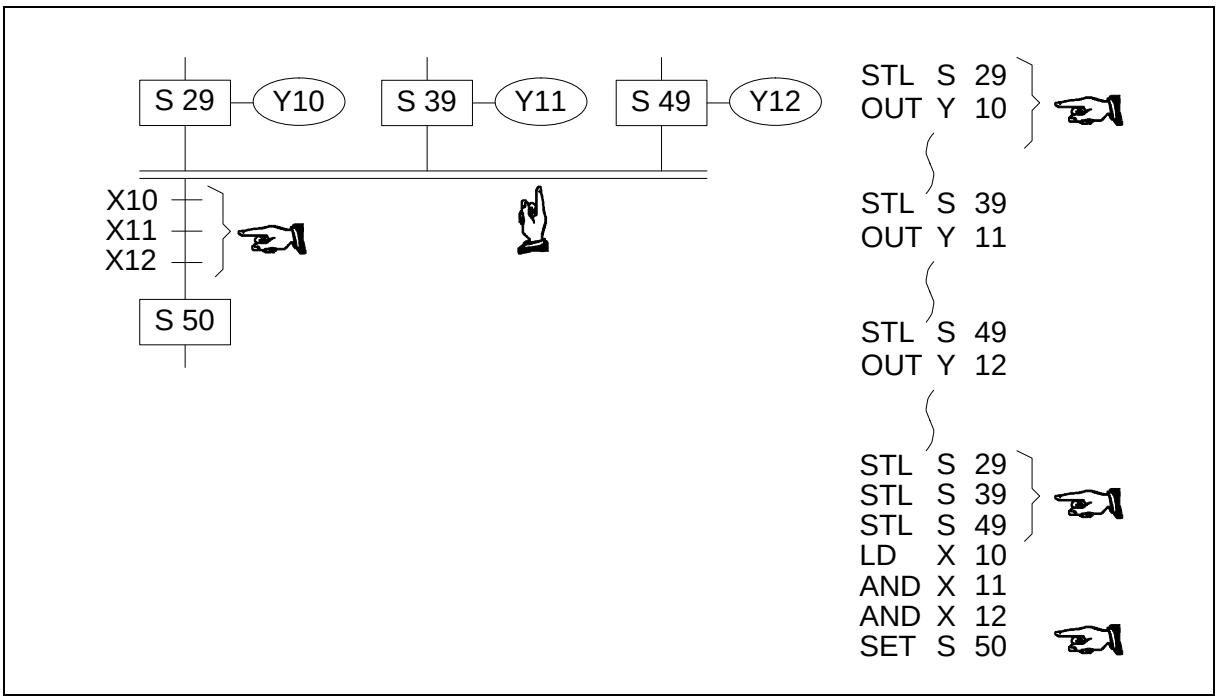


When a group of branch flows are activated, the user will often either;

a) 'Race' each flow against its counter parts. The flow which completes fastest would then activate a joining function ("First State Merge" described in the previous section) OR

b) The STL flow will not continue until ALL branch flows have completed there tasks. This is called a 'Multiple State Merge".

An explanation of Multiple State Merge now follows below.
In the example below, states S29, S39 and S49 must all be active. If the instruction list is viewed it can be seen that each of the states has its own operating/processing instructions but that also additional STL instructions have been linked together (in a similar concept as the basic AND instruction). Before state S50 can be activated the trigger conditions must also be active, in this example these are X10, X11 and X12. Once all states and input conditions are made the merging or joining state can be SET ON. As is the general case, all of the states used in the setting procedure are reset automatically.



Because more than one state is being simultaneously joined with further states (some times described as a parallel merge), a set of horizontal parallel lines are used to aid a quick visual recognition.

Limits on the number of branches

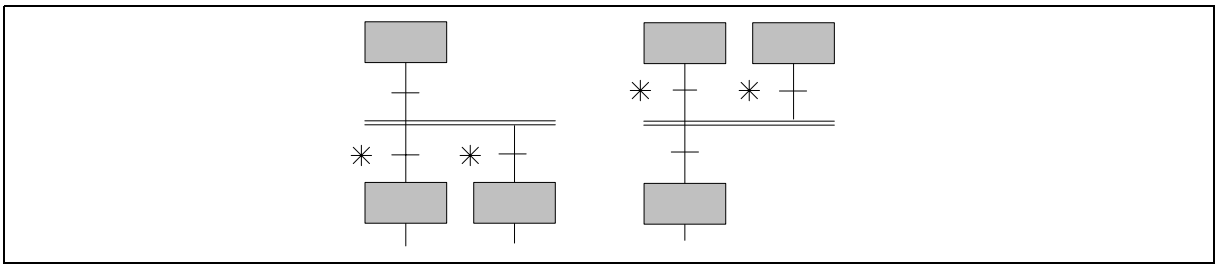
- Please see page 3-14 for general notes on programming STL branches.

Notes on using the FX-PCS/AT-EE software

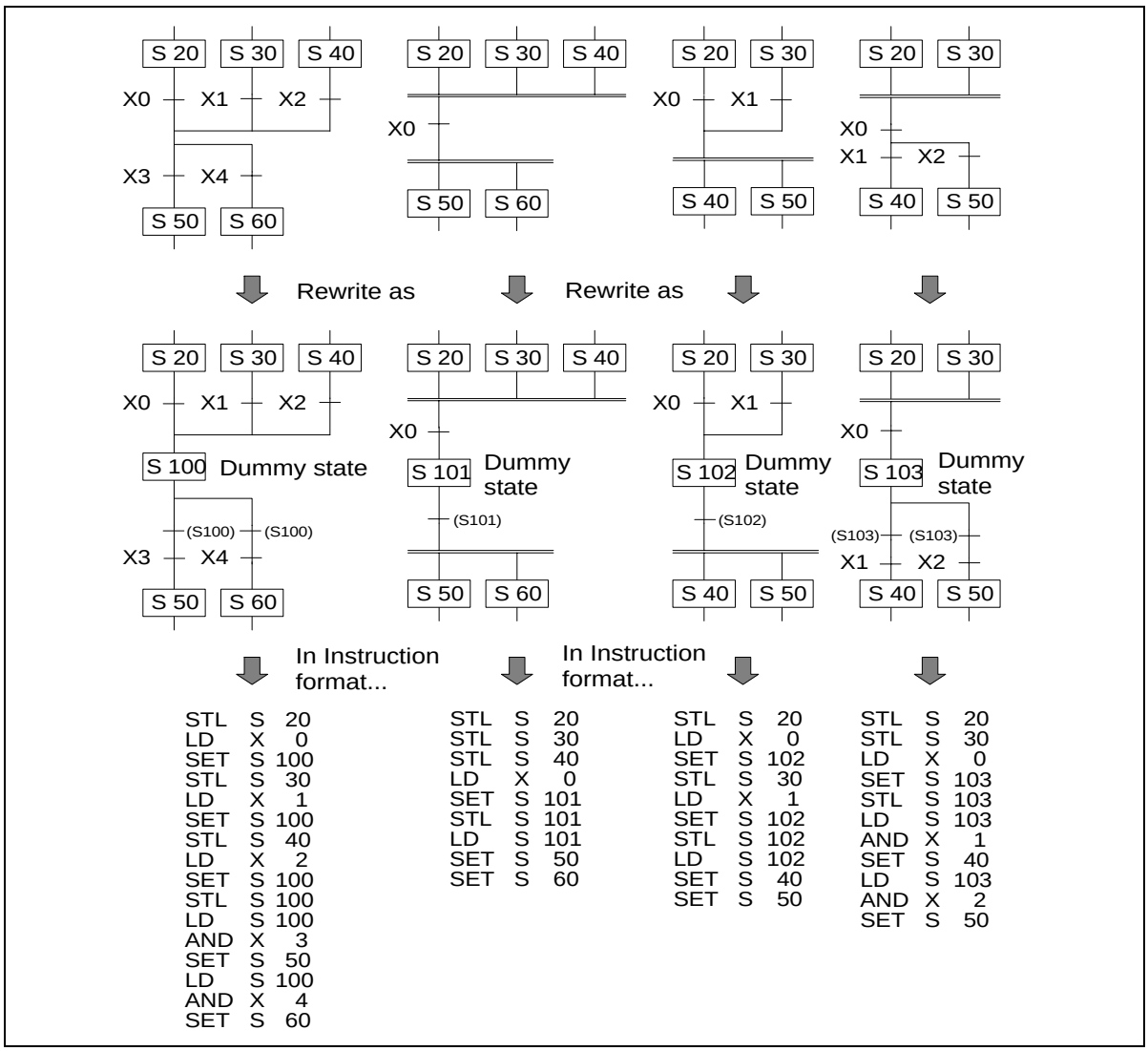
- Please see page 3-15 for precautions when using the FX-PCS-AT/EE software.

3.9 General Rules For Successful STL Branching

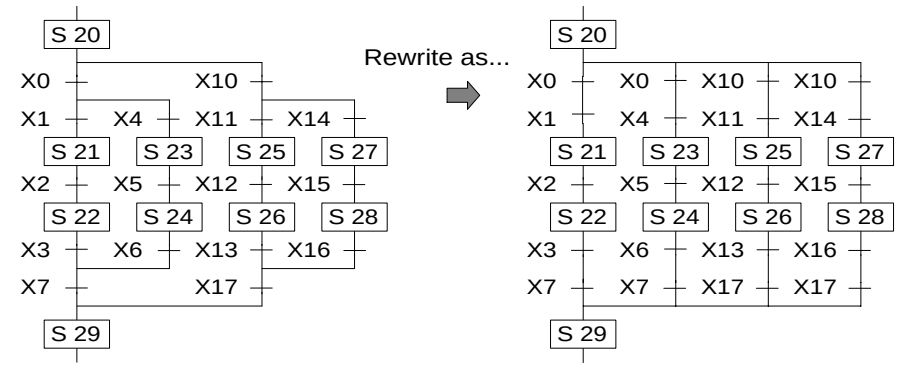
For each branch point 8 further branches may be programmed. There are no limits to the number of states contained in a single STL flow. Hence, the possibility exists for a single initial state to branch to 8 branch flows which in turn could each branch to a further 8 branch flows etc. If the programmable controllers program is read/written using instruction or ladder formats the above rules are acceptable. However, users of the FX-PCS/AT-EE programming package who are utilizing the STL programming feature are constrained by further restrictions to enable automatic STL program conversions (please see page 3-15 for more details).
When using branches, different types of branching /merging cannot be mixed at the same branch point. The item marked with a 'S' are transfer condition which are not permitted.



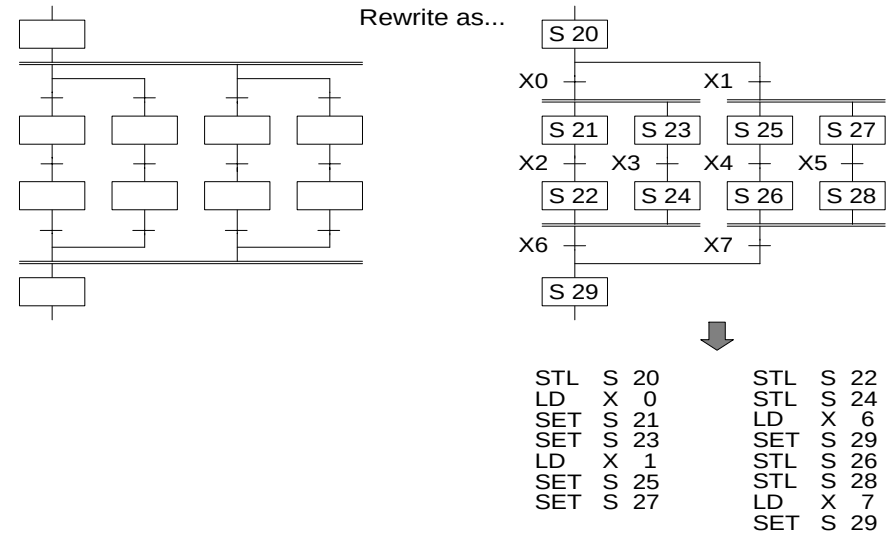
The following branch configurations/modifications are recommended:



Further recommended program changes:



Program violation!




```
STL  S  20        STL  S  22
LD   X   0        STL  S  24
SET  S  21        LD   X   6
SET  S  23        SET  S  29
LD   X   1        STL  S  26
SET  S  25        STL  S  28
SET  S  27        LD   X   7
                  SET  S  29
```

3.10 General Precautions When Using The FX-PCS/AT-EE Software

FX1S	FX1N	FX2N	FX2NC

This software has the ability to program in SFC flow diagrams. As part of this ability it can read and convert existing STL programs back into SFC flows even if they were never originally programmed using the FX-PCS/AT-EE software. As an aid to allowing this automatic SFC flow generation the following rules and points should be noted:

1) When an STL flow is started it should be initialized with one of the state devices from the range S0 to S9.

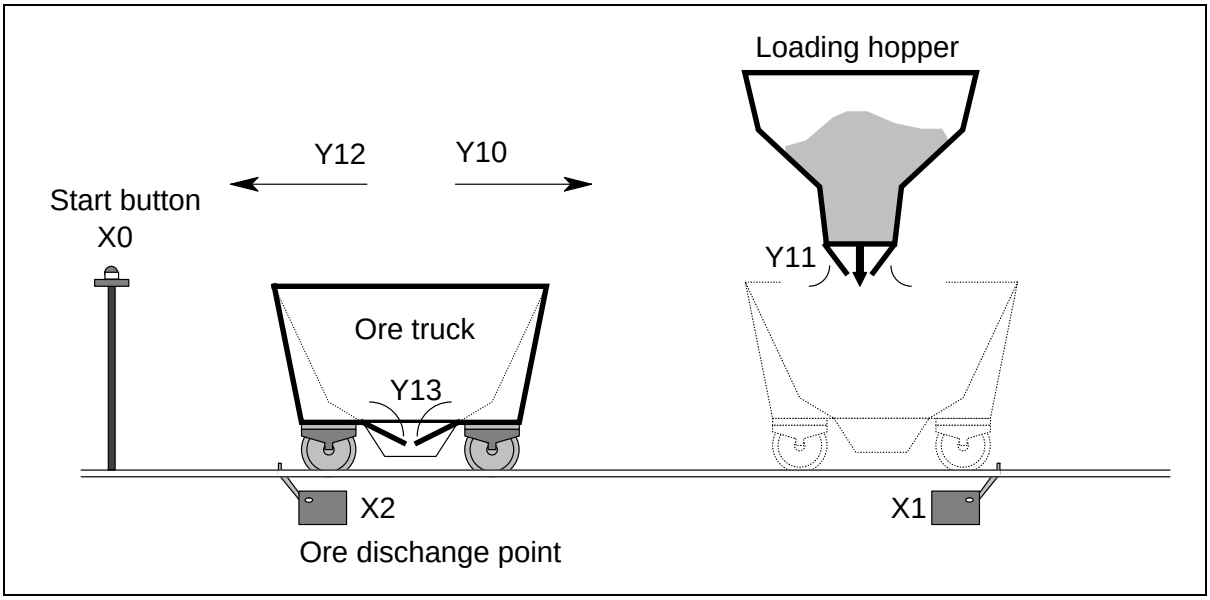
2) Branch selection or merging should always be written sequentially moving from left to right. This was demonstrated on page 3-11, i.e. on the selective branch S21 was specified before S31 which was specified before S41. The merge states were programmed in a similar manner, S29 proceeded S39 which proceeded S49.

3) The total number of branches which can be programmed with the STL programming mode are limited to a maximum of 16 circuits for an STL flow. Each branch point is limited to a maximum of 8 branching flows. This means two branch points both of 8 branch flows would equal the restriction. These restrictions are to ensure that the user can always view the STL flow diagram on the computer running the FX-PCS-AT/ EE software and that when it is needed, the STL program flow can be printed out clearly.

3.11 Programming Examples

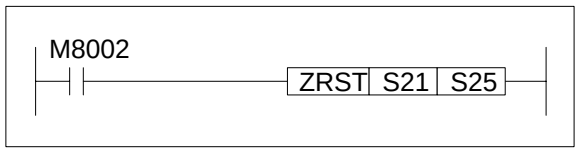
FX1S | FX1N | FX2N | FX2NC

3.11.1 A Simple STL Flow



This simple example is an excerpt from a semi-automatic loading-unloading ore truck program. This example program has a built in, initialization routine which occurs only when the PLC is powered from OFF to ON. This is achieved by using the special auxiliary relay M8002.

This activates a Zone ReSeT (ZRST is applied instruction 40) instruction which ensures all of the operational STL states within the program are reset. The program example opposite shows an M8002/ZRST example.

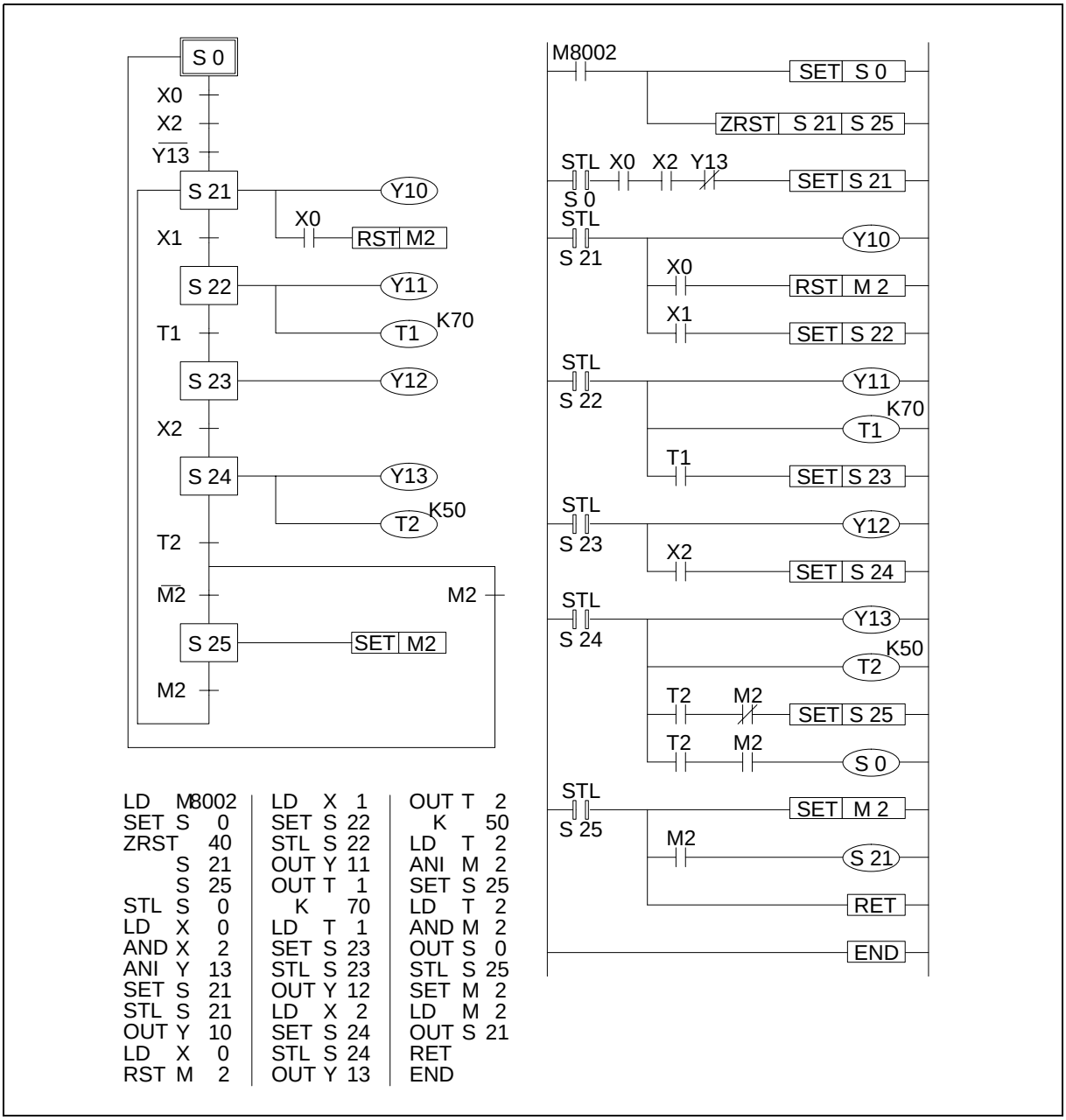


The push button X0 acts as a start button and a mode selection button. The STL state S0 is initialized with the ZRST instruction. The system waits until inputs X0 and X2 are given and Y 13 is not active. In the scenario this means the ore truck is positioned at the ore discharge point, i.e. above the position sensor X2. The ore truck is not currently discharging its load, i.e. the signal to open the trucks unloading doors (Y13) is not active and the start button (X0) has been given. Once all of the points have been met the program steps on to state S21.

On this state the ore cart is moved (Y10) and positioned (X1) at the loading hopper. If the start button (X0) is pressed during this stage the ore cart will be set into a repeat mode (M2 is reset) where the ore truck is immediately returned to the loading hopper after discharging its current load. This repeat mode must be selected on every return to the loading station.

Once at the loading point the program steps onto state S22. This state opens the hoppers doors (Y11) and fills the truck with ore. After a timed duration, state S23 is activated and the truck returns (Y12) to the discharge point (X2).

Once at the discharge point the truck opens its bottom doors (Y13). After a timed duration in which the truck empties its contents, the program checks to see if the repeat mode was selected on the last cycle, i.e. M2 is reset. If M2 was reset (in state S21) the program 'jumps' to step S21 and the ore truck is returned for immediate refilling. If M2 is not reset, i.e. it is active, the program cycles back to STL state S0 where the ore truck will wait until the start push button is given.
This is a simple program and is by no means complete but it identifies the way a series of tasks have been mapped to an STL flow.



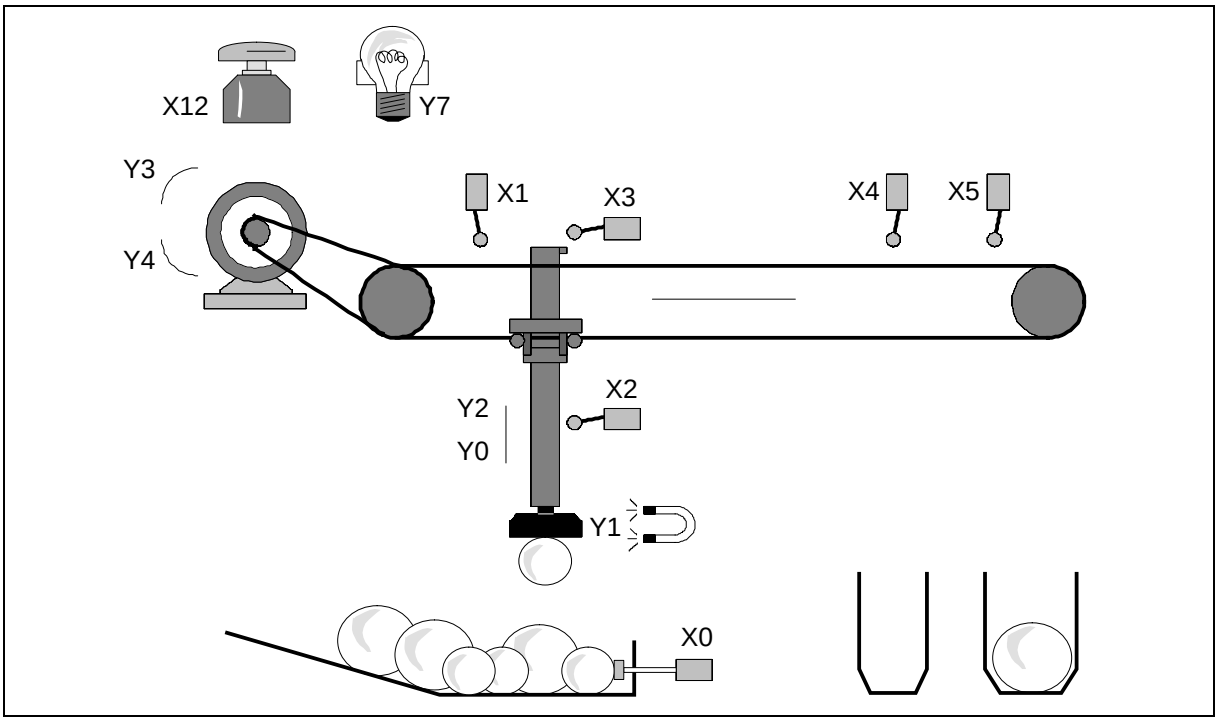

```
LD   M8002      LD   X   1      OUT T   2
SET  S   0      SET  S  22        K    50
ZRST     40     STL  S  22      LD   T   2
     S  21      OUT  Y  11      ANI  M   2
     S  25      OUT  T   1      SET  S  25
STL  S   0        K    70      LD   T   2
LD   X   0      LD   T   1      AND  M   2
AND  X   2      SET  S  23      OUT  S   0
ANI  Y  13      STL  S  23      STL  S  25
SET  S  21      OUT  Y  12      SET  M   2
STL  S  21      LD   X   2      LD   M   2
OUT  Y  10      SET  S  24      OUT  S  21
LD   X   0      STL  S  24      RET
RST  M   2      OUT  Y  13      END
```

Identification of normally closed contacts

This example has used the line convention to identify normally closed contacts, for further variations and different methods used to perform this task please see the information note page 3-3.

3.11.2 A Selective Branch/ First State Merge Example Program

The following example depicts an automatic sorting robot. The robot sorts two sizes of ball bearings from a mixed 'source pool' into individual storage buckets containing only one type of ball bearing.



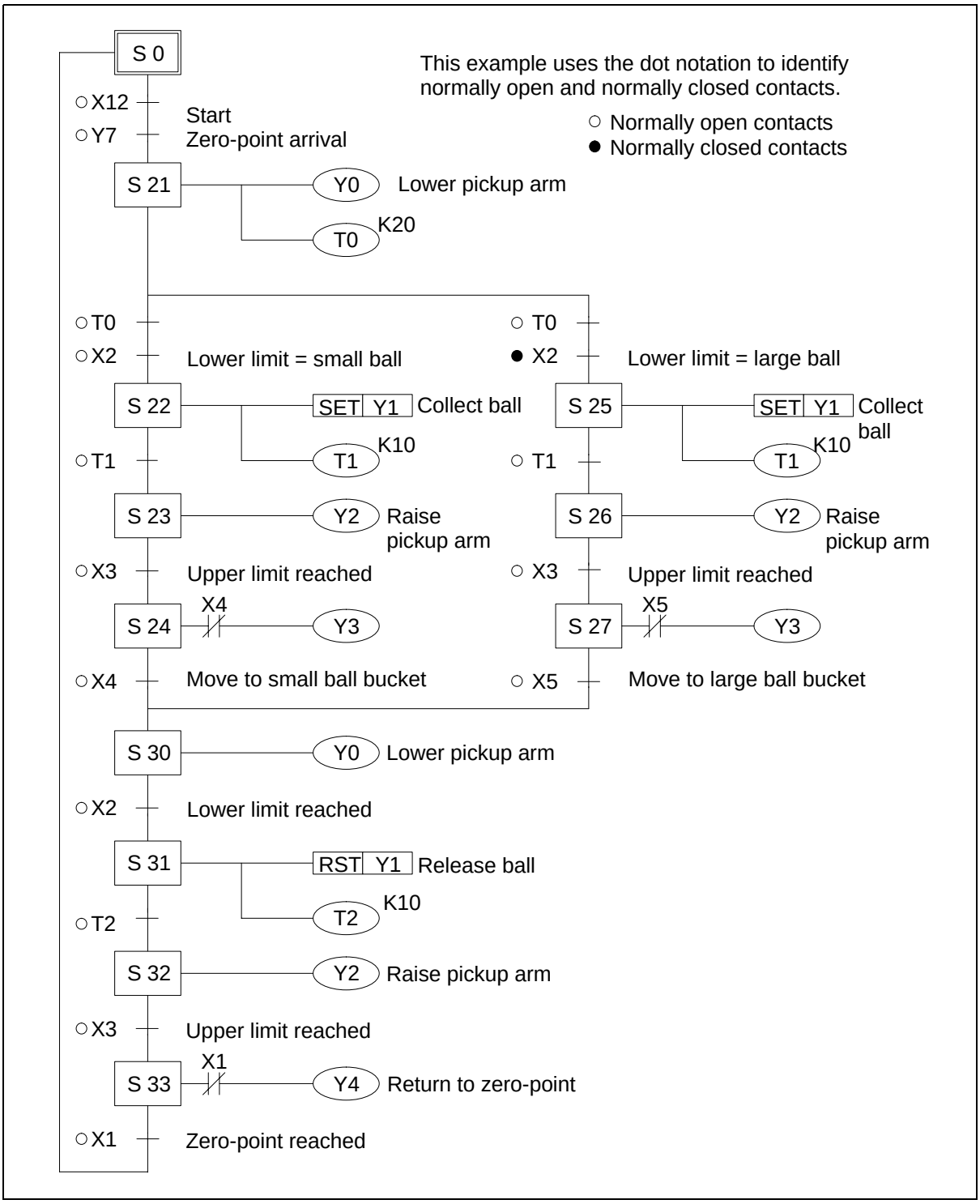
The sequence of physical events (from initial power On) are:

1) The pickup arm is moved to its zero-point when the start button (X12) is pressed. When the pickup arm reaches the zero-point the zero-point lamp (Y7) is lit.
2) The pickup arm is lowered (Y0) until a ball is collected (Y1). If the lower limit switch (X2) is made a small ball bearing has been collected; consequently no lower limit switch signal means a large ball bearing has been collected. Note, a proximity switch (X0) within the 'source pool' identifies the availability of ball bearings.
3) Depending on the collected ball, the pickup arm retracts (output Y2 is operated until X3 is received) and moves to the right (Y3) where it will stop at the limit switch (X4 or X5) indicating the container required for storage.
4) The program continues by lowering the pickup arm (Y0) until the lower limit switch (X2) is reached.
5) The collected ball being is released (Y1 is reset).
6) The pickup arm is retracted (Y2) once more.
7) The pickup arm is traversed back (Y4) to the zero-point (X1).

Points to note

- The Selective Branch is used to choose the delivery program for either small ball bearings or large ball bearings. Once the destination has been reached (i.e. step S24 or S27 has been executed) the two independent program flows are rejoined at step S30.
- The example program shown works on a single cycle, i.e. every time a ball is to be retrieved the start button (X12) must be pressed to initiate the cycle.

Full STL flow diagram/program.



3.12 Advanced STL Use

STL programming can be enhanced by using the Initial State Applied Instruction. This instruction has a mnemonic abbreviation of IST and a special function number of 60. When the IST instruction is used an automatic assignment of state relays, special auxiliary relays (M coils) is made. The IST instruction provides the user with a pre-formatted way of creating a multi-mode program. The modes available are:

a) Automatic:
   - Single step
   - Single cycle
   - Continuous

b) Manual:
   - Operator controlled
   - Zero return

More details on this instruction can be found on page 5-67.

Chapter Contents

4. Devices in Detail

FX1S | FX1N | FX2N | FX2NC

4.1 Inputs

Device Mnemonic: X

Purpose: Representation of physical inputs to the programmable controller (PLC)

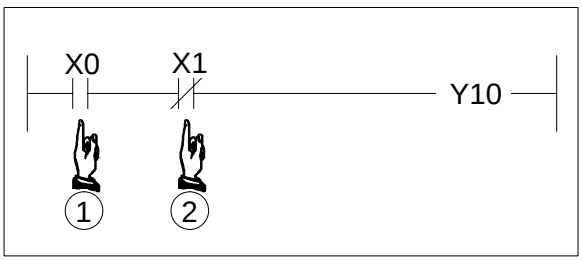
Alias: I/P
Inp
(X) Input
Input contact

Available forms: NO (①) and NC (②) contacts only
(see example device usage for references)

Devices numbered in: Octal, i.e. X0 to X7, X10 to X17

Further uses: None

Example device usage:



Available devices:

- Please see the information point on page 4-2, Outputs. Alternatively refer to the relevant tables for the selected PLC in chapter 8.

Configuration details:

- Please see chapter 9

4.2 Outputs

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: Y

Purpose: Representation of physical outputs from the programmable controller

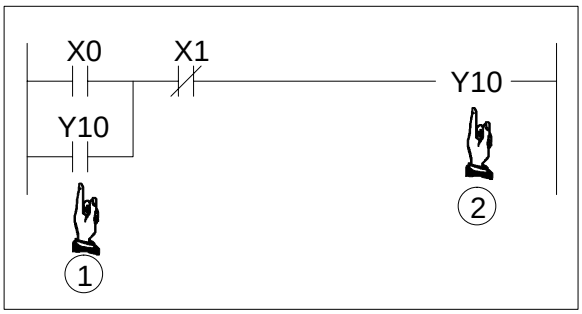
Alias: O/P
Otp
Out (Y)
Output (Y)
Output (coil/ relay/ contact)

Available forms: NO (①) and NC contacts and output coils (②)
(see example device usage for references)

Devices numbered in: Octal, i.e. Y0 to Y7, Y10 to Y17

Further uses: None

Example device usage:



Available devices:

PLC	Maximum number of inputs	Maximum number of outputs	Absolute total available I/O
FX1S	16	14	30
FX1N	128	128	128
FX2N	256 (addressable in software)	256 (addressable in software)	256 (Total addressed in software/hardware)
FX2NC			

- Please note, these are all the absolute maximums which are available. The values are subject to variations caused by unit selection. For configuration details please see chapter 9.
- For more information about the device availability for individual PLC's, please see chapter 8.

4.3 Auxiliary Relays

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: M

Purpose: Internal programmable controller status flag

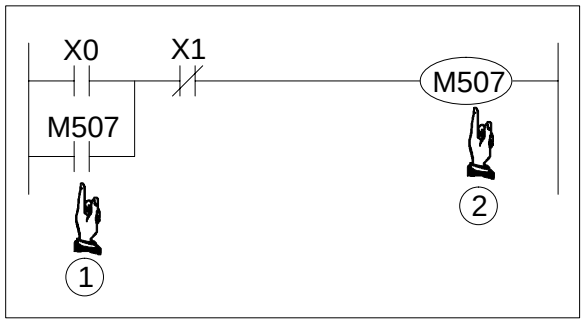
Alias: Auxiliary (coil/ relay/ contact/ flag)
M (coil/ relay/ contact /flag)
M (bit) device

Available forms: NO (①) and NC contacts and output coils (②)
(see example device usage for references)

Devices numbered in: Decimal, i.e. M0 to M9, M10 to M19

Further uses: General stable state auxiliary relays - see page 4-3
Battery backed/ latched auxiliary relays - see page 4-4
Special diagnostic auxiliary relays - see page 4-5

Example device usage:



4.3.1 General Stable State Auxiliary Relays

- A number of auxiliary relays are used in the PLC. The coils of these relays are driven by device contacts in the PLC in the same manner that the output relays are driven in the program.
  All auxiliary relays have a number of electronic NO and NC contacts which can be used by the PLC as required. Note that these contacts cannot directly drive an external load. Only output relays can be used to do this.

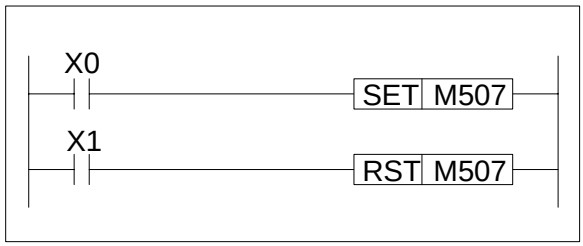
Available devices:

PLC	FX1S	FX1N	FX2N	FX2NC
General auxiliary relays	384 (M0 - 383)	384 (M0 - 383)	500 (M0 - 499)	500 (M0 - 499)
Battery backed/ latched relays	128 (M384 - 511)	1152 (M384 - 1535)	2572 (M500 - 3071)	2572 (M500 - 3071)
Total available	512	1536	3072	3072

- For more information about device availability for individual PLC's, please see chapter 8.

4.3.2 Battery Backed/ Latched Auxiliary Relays

There are a number of battery backed or latched relays whose status is retained in battery backed or EEPROM memory. If a power failure should occur all output and general purpose relays are switched off. When operation is resumed the previous status of these relays is restored.
The circuit shown on page 4-3 is an example of a self retaining circuit. Relay M507 is activated when X0 is turned ON. If X0 is turned OFF after the activation of M507, the ON status of M507 is self retained, i.e. the NO contact M507 drives the coil M507.
However, M507 is reset (turned OFF) when the input X1 is turned ON, i.e. the NC contact is broken.
A SET and RST (reset) instruction can be used to retain the status of a relay being activated momentarily.



External loads:

- Auxiliary relays are provided with countless number of NO contact points and NC contact points. These are freely available for use through out a PLC program. These contacts cannot be used to directly drive external loads. All external loads should be driven through the use of direct (Y) outputs.

4.3.3 Special Diagnostic Auxiliary Relays

A PLC has a number of special auxiliary relays. These relays all have specific functions and are classified into the following two types.

a) Using contacts of special auxiliary relays

- Coils are driven automatically by the PLC. Only the contacts of these coils may be used by a user defined program.

Examples: M8000: RUN monitor (ON during run)
M8002: Initial pulse (Turned ON momentarily when PLC starts)
M8012: 100 msec clock pulse

b) Driving coils of special auxiliary relays

- A PLC executes a predetermined specific operation when these coils are driven by the user.

Examples: M8033: All output statuses are retained when PLC operation is stopped
M8034: All outputs are disabled
M8039: The PLC operates under constant scan mode

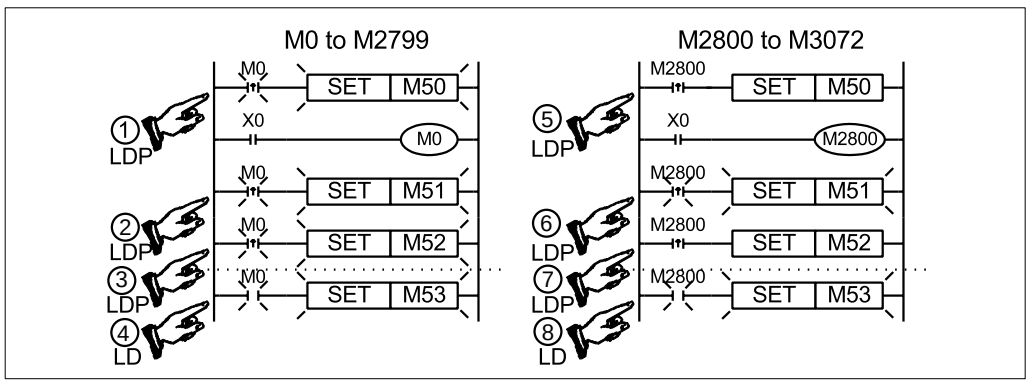
Available devices:

- Not all PLC's share the same range, quantity or operational meaning of diagnostic auxiliary relays. Please check the availability and function before using any device. PLC specific diagnostic ranges and meanings are available in chapter 6.

4.3.4 Special Single Operation Pulse Relays

FX1S	FX1N	FX2N	FX2NC

When used with the pulse contacts LDP, LDF, etc., M devices in the range M2800 to M3072 have a special meaning. With these devices, only the next pulse contact instruction after the device coil is activated.



Turning ON X0 causes M0 to turn ON.

- Contacts ①, ② and ③ are pulse contacts and activate for 1 scan.
- Contact ④ is a normal LD contact and activates while M0 is ON.

Turning ON X0 causes M2800 to turn ON.

- Contact ⑥ is a pulse contact and activates for 1 scan.
- Contacts ⑤ and ⑦ are pulse contacts of the same M device as contact ⑥. Contact ⑥ has already operated, so contact ⑤ and ⑦ do not operate.
- Contact ⑧ is a normal LD contact and activates while M2800 is ON.

4.4 State Relays

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: S

Purpose: Internal programmable controller status flag

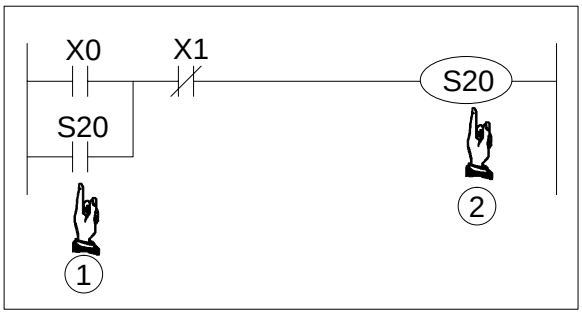
Alias: State (coil/ relay/ contact/ flag)
S (coil/ relay/ contact /flag)
STL step (coil/ relay/ contact /flag)
Annunciator flag

Available forms: NO (①) and NC contacts and output coils (②)
(see example device usage for references)

Devices numbered in: Decimal, i.e. S0 to S9, S10 to S19

Further uses: General stable state - state relays - see page 4-6
Battery backed/ latched state relays - see page 4-7
STL step relays - see page 4-8
Annunciator flags - see page 4-9

Example device usage:



4.4.1 General Stable State - State Relays

A number of state relays are used in the PLC. The coils of these relays are driven by device contacts in the PLC in the same manner that the output relays are driven in the program.
All state relays have a number of electronic NO and NC contacts which can be used by the PLC as required. Note that these contacts cannot directly drive an external load. Only output relays can be used to do this.

Available devices:

- Please see the information point on page 4-7 ‘Battery backed/ latched state relays’, or see the relevant tables for the selected PLC in chapter 8.

4.4.2 Battery Backed/ Latched State Relays

There are a number of battery backed or latched relays whose status is retained in battery backed or EEPROM memory. If a power failure should occur all output and general purpose relays are switched off. When operation is resumed the previous status of these relays is restored.

Available devices:

PLC	FX1S	FX1N	FX2N	FX2NC
General state relays	N/A	N/A	500 (S0 - 499)	
Battery backed/ latched relays	128 (S0 - 127)	1000 (S0 - 999)	500 (S500 - 999)	
Total available	128	1000	1000	

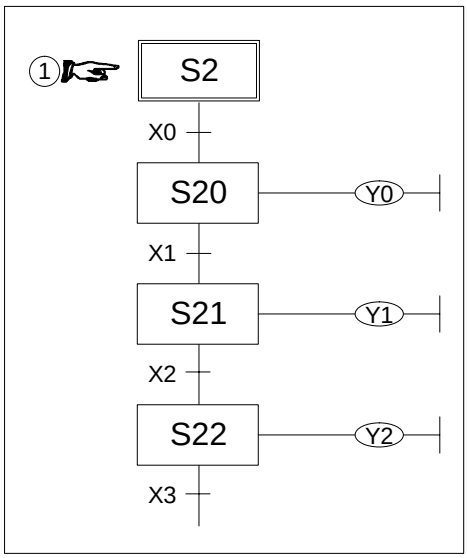
- For more information about device availability for individual PLC's, see chapter 8.

External loads:

- State relays are provided with countless number of NO contact points and NC contact points, and are freely available for use through out a PLC program. These contacts cannot be used to directly drive external loads. All external loads should be driven through the use of direct (ex. Y) outputs.

4.4.3 STL Step Relays

States (S) are very important devices when programming step by step process control. They are used in combination with the basic instruction STL.
When all STL style programming is used certain states have a pre-defined operation. The step identified as ① in the figure opposite is called an 'initial state'. All other state steps are then used to build up the full STL function plan. It should be remembered that even though remaining state steps are used in an STL format, they still retain their general or latched operation status. The range of available devices is as specified in the information point of the previous section.



Assigned states:

- When the applied instruction IST (Initial STate function 60) is used, the following state devices are automatically assigned operations which cannot be changed directly by a users program:

S0	: Manual operation initial state
S1	: Zero return initial state
S2	: Automatic operation initial state
S10 to S19	: Allocated for the creation of the zero return program sequence

Monitoring STL programs:

- To monitor the dynamic-active states within an STL program, special auxiliary relay M8047 must be driven ON.

STL/SFC programming:

- For more information on STL/SFC style programming, please see chapter 3.

IST instruction:

- For more information on the IST instruction please see page 5-67.

4.4.4 Annunciator Flags

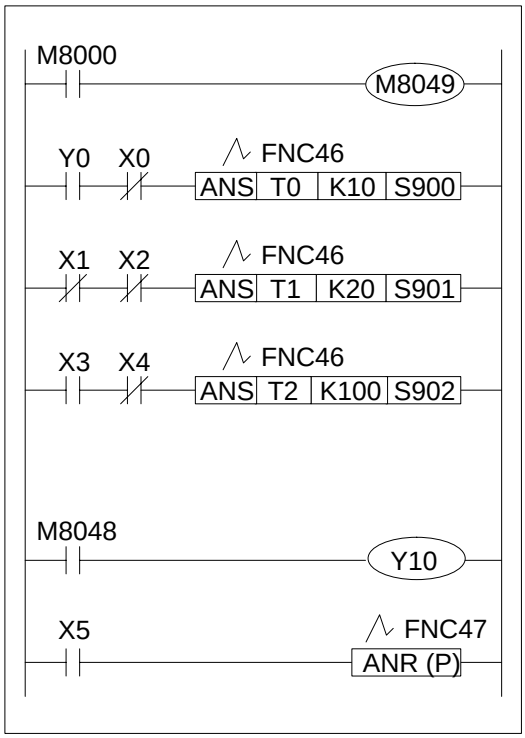
FX1S | FX1N | FX2N | FX2NC

Some state flags can be used as outputs for external diagnosis (called annunciation) when certain applied instructions are used. These instructions are;

ANS function 46: ANnunciator Set - see page 5-47
ANR function 47: ANnunciator Reset - see page 5-47

When the annunciator function is used the controlled state flags are in the range S900 to S999 (100 points). By programming an external diagnosis circuit as shown below, and monitoring special data register D8049, the lowest activated state from the annunciator range will be displayed.
Each of the states can be assigned to signify an error or fault condition. As a fault occurs the associated state is driven ON. If more than one fault occurs simultaneously, the lowest fault number will be displayed. When the active fault is cleared the next lowest fault will then be processed.
This means that for a correctly prioritized diagnostic system the most dangerous or damaging faults should activate the lowest state flags, from the annunciator range. All state flags used for the annunciator function fall in the range of battery backed/ latched state registers.

Monitoring is enabled by driving special auxiliary relay M8049 ON.
State S900 is activated if input X0 is not driven within one second after the output Y0 has been turned ON.
State S901 is activated when both inputs X1 and X2 are OFF for more than two seconds.
If the cycle time of the controlled machine is less than ten seconds, and input X3 stays ON, state S902 will be set ON if X4 is not activated within this machine cycle time.
If any state from S900 to S999 is activated, i.e. ON, special auxiliary relay M8048 is activated to turn on failure indicator output Y10.
The states activated by the users error / failure diagnosis detection program, are turned OFF by activating input X5. Each time X5 is activated, the active annunciator states are reset in ascending order of state numbers.



4.5 Pointers

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: P

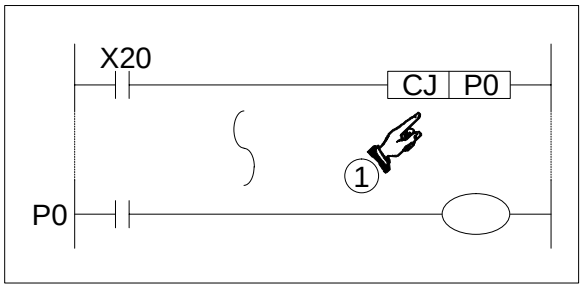
Purpose: Program flow control
Alias: Pointer
Program pointer
P

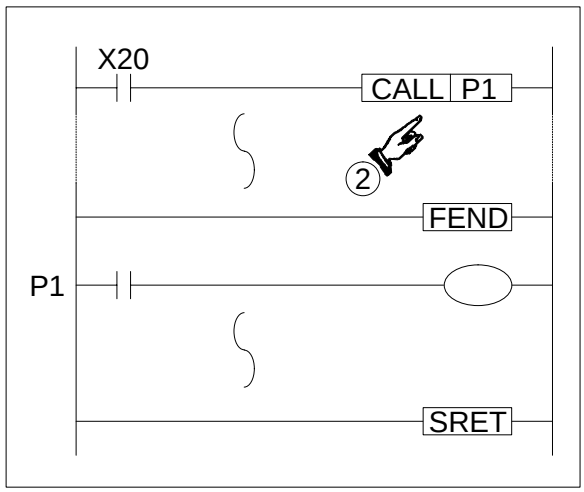
Available forms: Label: appears on the left of the left hand bus bar when the program is viewed in ladder mode.

Devices numbered in: Decimal, i.e. P0 to P9, P10 to P19

Further uses: Can be used with conditional jump statements (CJ function 00)
- see page 5-5 and item ① on the example device usage diagram.
Can be used with call statements
- see page 5-7 and item ② on the example device usage diagram

Example device usage:





Available devices:

- FX1S PLC's have 64 pointers; available from the range of P0 to P63.
- FX1N, FX2N and FX2NC PLC's have 128 pointers; available from the range of P0 to P127.

Jumping to the end of the program:

- When using conditional jump instructions (CJ, function 00) the program end can be jumped to automatically by using the pointer P63 within the CJ instruction. Labelling the END instruction with P63 is not required.

Device availability:

- For more information about device availability for individual PLC's, please see chapter 8.

4.6 Interrupt Pointers

FX1S	FX1N	FX2N	FX2NC

Device Mnemonic: I

Purpose: Interrupt program marker

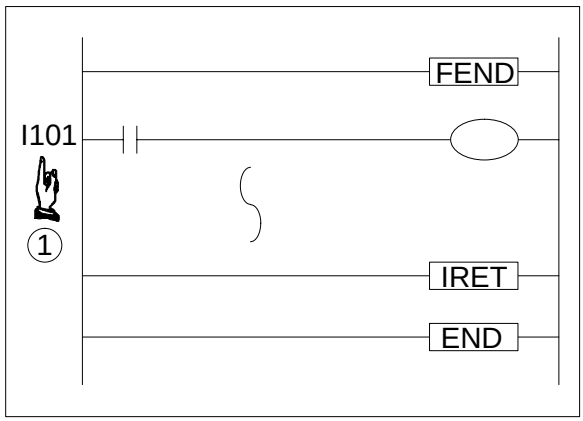
Alias: Interrupt
High speed interrupt
I

Available forms: Label: appears on the left of the left hand bus bar when the program is viewed in ladder mode
(see ① in the example device usage diagram).

Devices numbered in: Special numbering system based on interrupt device used and input triggering method

Further uses: Input interrupts - see page 4-12
Timer interrupts - see page 4-12
Disabling interrupts - see page 4-13
Counter interrupts - see page 4-13

Example device usage:



Additional applied instructions:

- Interrupts are made up of an interrupt device, an interrupt pointer and various usage of three, dedicated interrupt applied instructions;

  - IRET function 03: interrupt return - see page 5-9
  - EI function 04: enable interrupt - see page 5-9
  - DI function 05: disable interrupt - see page 5-9

Nested levels:

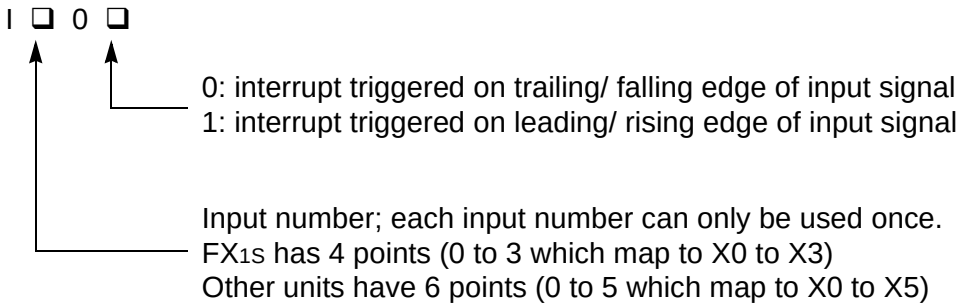
- While an interrupt is processing all other interrupts are disabled. To achieve nested interrupts the EI-DI instruction must be programmed within an interrupt routine. Interrupts can be nested for two levels.

Pointer position:

- Interrupt pointers may only be used after an FEND instruction (first end instruction, function 06).

4.6.1 Input Interrupts

Identification of interrupt pointer number:



Example: I001

The sequence programmed after the label (indicated by the I001 pointer) is executed on the leading or rising edge of the input signal X0. The program sequence returns from the interruption program when an IRET instruction is encountered.

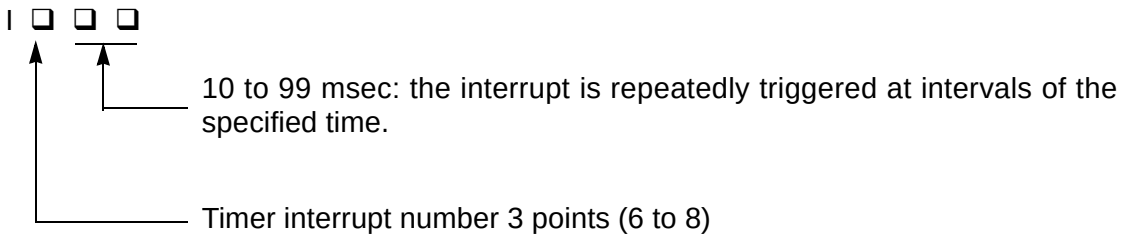
Rules of use:

- The following points must be followed for an interrupt to operate;
  - Interrupt pointers cannot have the same number in the '100's' position, i.e. I100 and I101 are not allowed.
  - The input used for the interrupt device must not coincide with inputs already allocated for use by other high speed instructions within the user program.

4.6.2 Timer Interrupts

FX1S | FX1N | FX2N | FX2NC

Identification of interrupt pointer number:



Example: I610

The sequence programmed after the label (indicated by the I610 pointer) is executed at intervals of 10msec. The program sequence returns from the interruption program when an IRET instruction is encountered.

Rules of use:

- The following points must be followed for an interrupt to operate;
  - Interrupt pointers cannot have the same number in the '100's' position, i.e. I610 and I650 are not allowed.

4.6.3 Disabling Individual Interrupts

Individual interrupt devices can be temporarily or permanently disabled by driving an associated special auxiliary relay. The relevant coils are identified in the tables of devices in chapter 6. However for all PLC types the head address is M8050, this will disable interrupt I0❑❑.

Driving special auxiliary relays:

- Never drive a special auxiliary coil without first checking its use. Not all PLC's assign the same use to the same auxiliary coils.

Disabling high speed counter interrupts

- These interrupts can only be disabled as a single group by driving M8059 ON. Further details about counter interrupts can be found in the following section.

4.6.4 Counter Interrupts

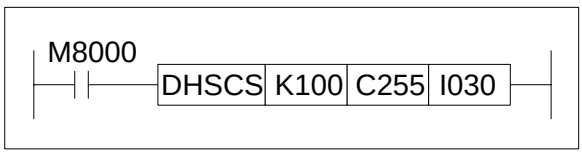
FX1S | FX1N | FX2N | FX2NC

Identification of interrupt pointer number:

I 0 ❑ 0

Counter interrupt number 6 points (1 to 6). Counter interrupts can be entered as the output devices for High Speed Counter Set (HSCS, FNC 53). To disable the Counter Interrupts Special Auxiliary Relay M8059 must be set ON.

Example:



The sequence programmed after the label (indicated by the I030 pointer) is executed once the value of High Speed Counter C255 reaches/equals the preset limit of K100 identified in the example HSCS.

Additional notes:

- Please see the following pages for more details on the HSSC applied instruction.
  - High Speed Counter Set, HSCS FNC 53 - see page 5-55

4.7 Constant K

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: K

Purpose: Identification of constant decimal values

Alias: Constant
K (value/ constant)
K

Available forms: Numeric data value, when used for 16bit data, values can be selected from the range -32,768 to +32,767
For 32bit data, values from the range -2,147,483,648 to + 2,147,483,647 can be used.

Devices numbered in: N/A. This device is a method of local instruction data entry. There is no limit to the number of times it can be used.

Further uses: K values can be used with timers, counters and applied instructions

Example device usage: N/A

4.8 Constant H

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: H

Purpose: Identification of constant hexadecimal values

Alias: Constant
H (value/ constant)
Hex (value/ constant)
H

Available forms: Alpha-numeric data value, i.e. 0 to 9 and A to F (base 16).
When used for 16bit data, values can be selected from the range 0 to FFFF.
For 32bit data, values from the range 0 to FFFFFFFF can be used.

Devices numbered in: N/A. This device is a method of local instruction data entry. There is no limit to the number of times it can be used.

Further uses: Hex values can be used with applied instructions

Example device usage: N/A

4.9 Timers

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: T

Purpose: Timed durations

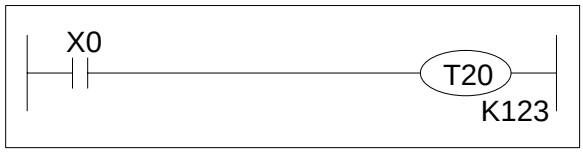
Alias: Timer(s)
T

Available forms: A driven coil sets internal PLC contacts (NO and NC contacts available). Various timer resolutions are possible, from 1 to 100 msec, but availability and quantity vary from PLC to PLC. The following variations are also available:-
Selectable timer resolutions - see page 4-16
Retentive timers - see page 4-17
Timers used in interrupt and 'CALL' subroutines - see page 4-18

Devices numbered in: Decimal, i.e T0 to T9, T10 to T19.

Further uses: None

Example device usage:



Available devices:

Timer Resolution	FX1S	FX1N	FX2N	FX2NC
100 msec	63 (T0 - 62)	200 (T0 - 199)		
10 msec	\ 31 (T32 - 62)	46 (T200 - 245)		
1 msec	1 (T63)	N/A		
Retentive 1 msec	N/A	4 (T246 - 249)		
Retentive 100 msec	N/A	6 (T250 - 255)		

\ Selectable timers taken from the main range of 100 msec timers, see page 4-16.

Timer accuracy:

- See page 4-18.

4.9.1 General timer operation

Timers operate by counting clock pulses (1, 10 and 100 msec). The timer output contact is activated when the count data reaches the value set by the constant K. The overall duration or elapsed time, for a timers operation cycle, is calculated by multiplying the present value by the timer resolution, i.e.

A 10 msec timer with a present value of 567 has actually been operating for:

567× 10 msec
567× 0.01 sec = 5.67 seconds

Timers can either be set directly by using the constant K to specify the maximum duration or indirectly by using the data stored in a data register (ex. D). For the indirect setting, data registers which are battery backed/ latched are usually used; this ensures no loss of data during power down situations. If however, the voltage of the battery used to perform the battery backed service, reduces excessively, timer malfunctions may occur.

4.9.2 Selectable Timers

FX1S	FX1N	FX2N	FX2NC

On certain programmable controllers, driving a special auxiliary coil redefines approximately half of the 100 msec timers as 10 msec resolution timers. The following PLC's and timers are subject to this type of selection.

- For FX1S, driving M8028 ON, timers T32 to 62 (31 points) are changed to 10 msec resolution.

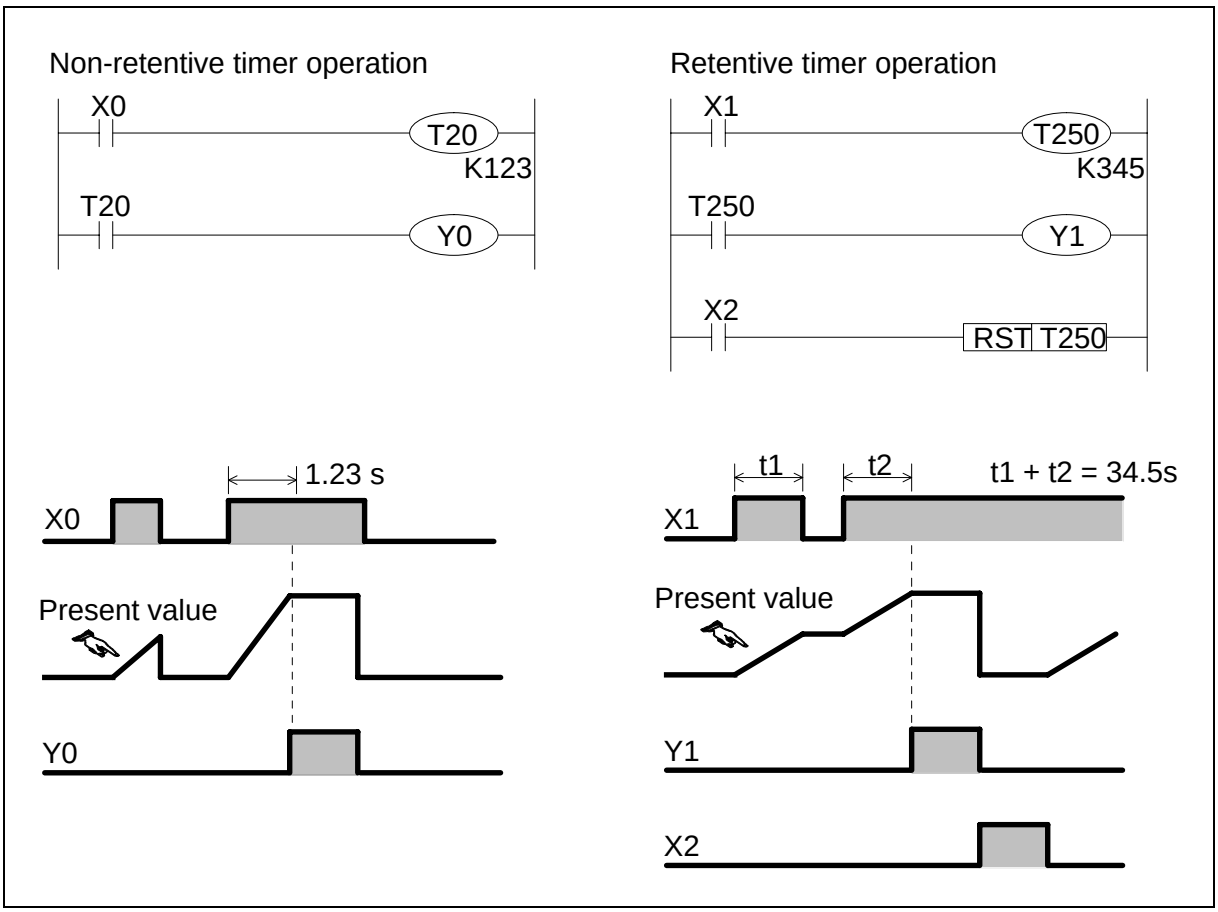
Driving special auxiliary coils:

- Please check the definition of special auxiliary coils before using them.
  Not all PLC's associate the same action to the same device.

4.9.3 Retentive Timers

FX1S | FX1N | FX2N | FX2NC

A retentive timer has the ability to retain the currently reached present value even after the drive contact has been removed. This means that when the drive contact is re-established a retentive timer will continue from where it last reached.
Because the retentive timer is not reset when the drive contact is removed, a forced reset must be used. The following diagram shows this in a graphical format.



Using timers in interrupt or 'CALL' subroutines:

- Please see page 4-18.

Available devices:

- Please see the information table on page 4-15.

4.9.4 Timers Used in Interrupt and 'CALL' Subroutines

FX1S | FX1N | FX2N | FX2NC

If timers T192 to T199 and T246 to T249 are used in a CALL subroutine or an interruption routine, the timing action is updated at the point when an END instruction is executed. The output contact is activated when a coil instruction or an END instruction is processed once the timers current value has reached the preset (maximum duration) value.
Timers other than those specified above cannot function correctly within the specified circumstances.
When an interrupt timer (1 msec resolution) is used in an interrupt routine or within a 'CALL' subroutine, the output contact is activated when the first coil instruction of that timer is executed after the timer has reached its preset (maximum duration) value.

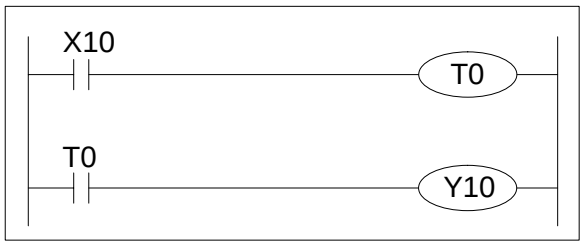
4.9.5 Timer Accuracy

Timer accuracy can be affected by the program configuration. That is to say, if a timer contact is used before its associated coil, then the timer accuracy is reduced.
The following formulas give maximum and minimum errors for certain situations.
However, an average expected error would be approximately;

1.5 × The program scan time

Condition 1:

The timer contact appears after the timer coil.



Maximum timing error:

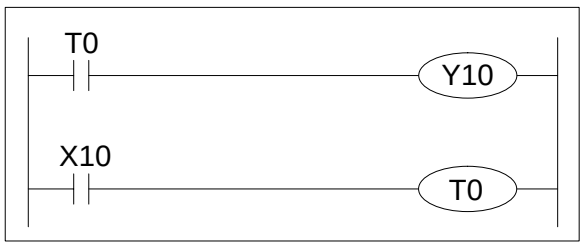
2 × Scan time + The input filter time

Minimum timing error:

Input filter time - The timer resolution

Condition 2:

The timer contact appears before the timer coil.



Maximum timing error:

3 × Scan time + The input filter time

Minimum timing error:

Input filter time- The timer resolution

Internal timer accuracy:

- The actual accuracy of the timing elements within the PLC hardware is; ± 10 pulses per million pulses. This means that if a 100 msec timer is used to time a single day, at the end of that day the timer will be within 0.8 seconds of the true 24 hours or 86,400 seconds. The timer would have processed approximately 864,000; 100 msec pulses.

4.10 Counters

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: C

Purpose: Event driven delays

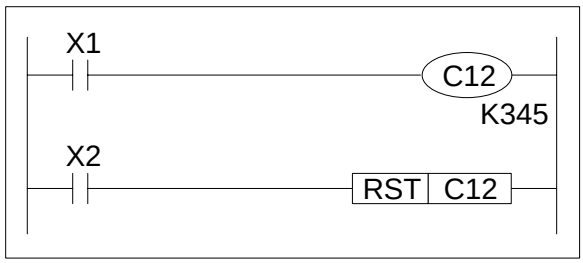
Alias: Counter(s)
C

Available forms: A driven coil sets internal PLC contacts (NO and NC contacts available).
Various counter resolutions are possible including;
General/latched 16bit up counters - see page 4-20
General/latched 32bit bi-directional counters - see page 4-21
(The availability and use of all these counters is PLC specific - please check availability before use)

Devices numbered in: Decimal, i.e C0 to C9, C10 to C19

Further uses: None

Example device usage:



Available devices:

Counter Resolution	FX1S	FX1N	FX2N	FX2NC
General 16bit up counter	16 (C0 - 15)	16 (C0 - 15)	100 (C0 - 99)	
Latched 16bit up counter	16 (C16 - 31)	184 (C16 - 199)	100 (C100 - 199)	
General 32bit bi-directional counter	N/A	20 (C200 - 219)		
Latched 32bit bi-directional counter	N/A	15 (C220 - 234)		

High speed counters:

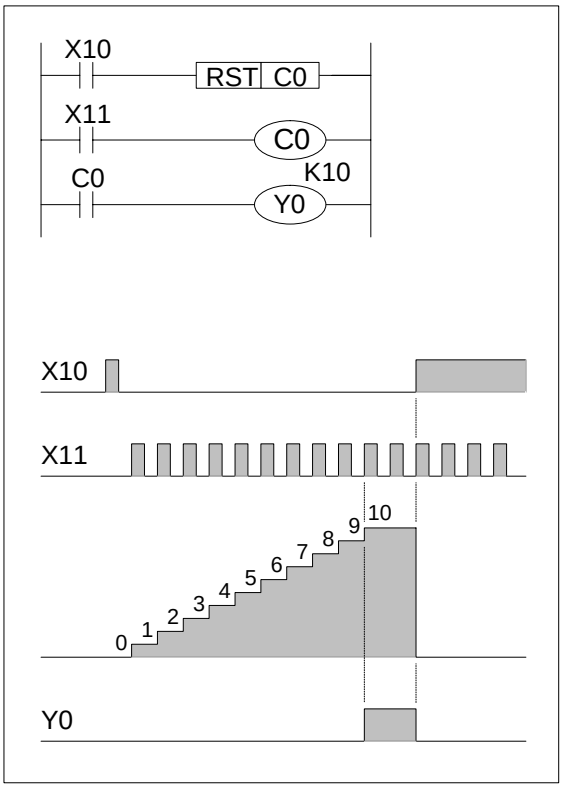
- For high speed counters please see page 4-22.

Setting ranges for counters:

- 16bit and 32bit up counters:1 to +32,767
- 32bit bi-directionalcounters: -2,147,483,648 to +2,147,483,647

4.10.1 General/ Latched 16bit UP Counters

The current value of the counter increases each time coil C0 is turned ON by X11. The output contact is activated when the coil is turned ON for the tenth time (see diagram). After this, the counter data remains unchanged when X11 is turned ON. The counter current value is reset to '0' (zero) when the RST instruction is executed by turning ON X10 in the example. The output contact Y0 is also reset at the same time.
Counters can be set directly using constant K or indirectly by using data stored in a data register (ex. D). In an indirect setting, the designation of D10 for example, which contains the value "123" has the same effect as a setting of "K123".
If a value greater than the counter setting is written to a current value register, the counter counts up when the next input is turned ON. This is true for all types of counters.
Generally, the count input frequency should be around several cycles per second.



Battery backed/latched counters:

- Counters which are battery backed/ latched are able to retain their status information, even after the PLC has been powered down. This means on re-powering up, the latched counters can immediately resume from where they were at the time of the original PLC power down.

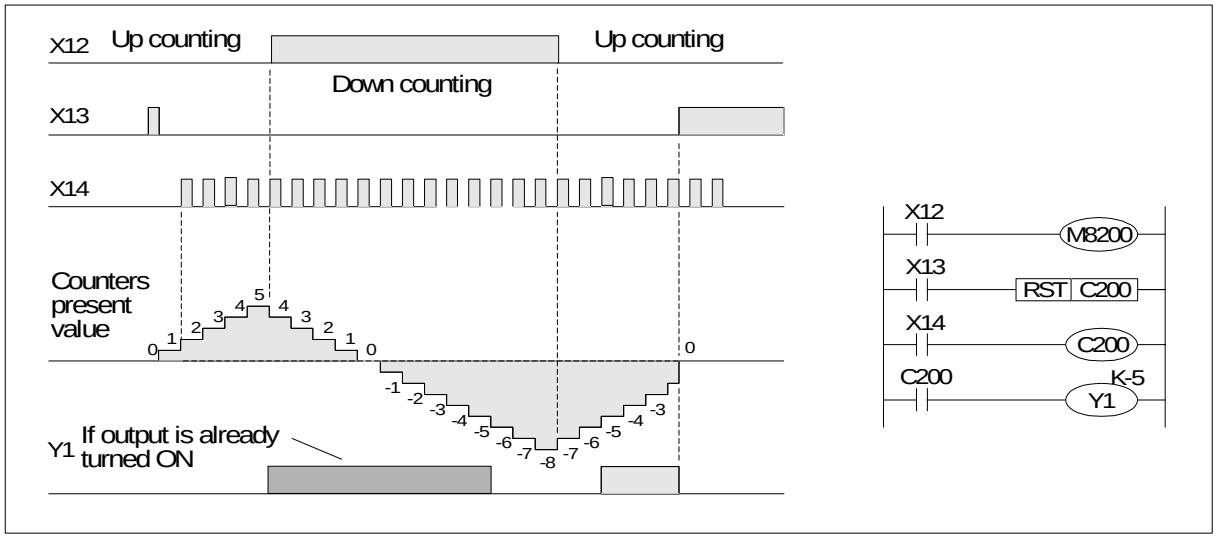
Available devices:

- Please see the information table on page 4-19.

4.10.2 General/ Latched 32bit Bi-directional Counters

FX1S | FX1N | FX2N | FX2NC

The counter shown in the example below, activates when its coil is driven, i.e. the C200 coil is driven. On every occasion the input X14 is turned from OFF to ON the current value or current count of C200 is incremented.



The output coil of C200 is set ON when the current value increases from “-6” to “-5”. However, if the counters value decreases from “-5” to “-6” the counter coil will reset. The counters current value increases or decreases independently of the output contact state (ON/OFF). Yet, if a counter counts beyond +2,147,483,647 the current value will automatically change to -2,147,483,648. Similarly, counting below -2,147,483,648 will result in the current value changing to +2,147,483,647. This type of counting technique is typical for “ring counters”. The current value of the active counter can be rest to "0" (zero) by forcibly resetting the counter coil; in the example program by switching the input X13 ON which drives the RST instruction. The counting direction is designated with special auxiliary relays M8200 to M8234.

Battery backed/ latched counters:

- Counters which are battery backed/ latched are able to retain their status information, even after the PLC has been powered down. This means on re-powering up, the latched counters can immediately resume from where they were at the time of the original PLC power down.

Available devices:

- Please see the information table on page 4-19.

Selecting the counting direction:

- If M8☆☆☆ for C☆☆☆ is turned ON, the counter will be a down counter. Conversely, the counter is an up counter when M8☆☆☆ is OFF.

4.11 High Speed Counters

FX1S	FX1N	FX2N	FX2NC

Device Mnemonic: C

Purpose: High speed event driven delays

Alias: Counter (s)
C
High speed counter (s)
Phase counters

Available forms: A driven coil sets internal PLC contacts (NO and NC contacts available). There are various types of high speed counter available but the quantity and function vary from PLC to PLC. Please check the following sections for device availability;

FX1S and FX1N - see page 4-24
FX2N and FX2NC - see page 4-25

The following sections refer to counter types;
1 phase counters (user start and reset) - see page 4-29
1 phase counters (assigned start and reset) - see page 4-30
2 phase bi-directional counters - see page 4-31
A/B phase counters - see page 4-32

Devices numbered in: Decimal, i.e C235 to C255

Further uses: None

Example device usage: For examples on each of the available forms please see the relevant sections.

Basic high speed counter operation:

- For information on basic high speed counters please see page 4-23.

4.11.1 Basic High Speed Counter Operation

Although counters C235 to C255 (21 points) are all high speed counters, they share the same range of high speed inputs. Therefore, if an input is already being used by a high speed counter, it cannot be used for any other high speed counters or for any other purpose, i.e as an interrupt input.
The selection of high speed counters are not free, they are directly dependent on the type of counter required and which inputs are available.

Available counter types;

a) 1 phase with user start/reset: C235 to C240

b) 1 phase with assigned start/reset: C241 to C245

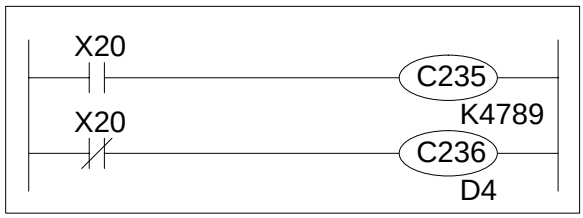
c) 2 phase bi-directional: C246 to C250

d) A/B phase type: C251 to C255

Please note ALL of these counters are 32bit devices.
High speed counters operate by the principle of interrupts. This means they are event triggered and independent of cycle time. The coil of the selected counter should be driven continuously to indicate that this counter and its associated inputs are reserved and that other high speed processes must not coincide with them.

Example:

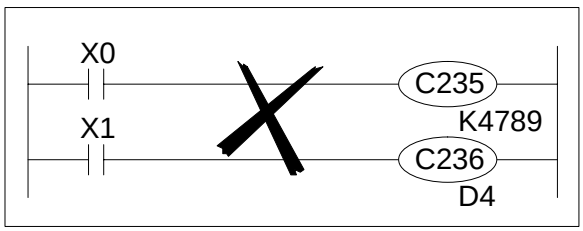
When X20 is ON, high speed counter C235 is selected. The counter C235 corresponds to count input X0. X20 is NOT the counted signal. This is the continuous drive mentioned earlier. X0 does not have to be included in the program. The input assignment is hardware related and cannot be changed by the user.



When X20 is OFF, coil C235 is turned OFF and coil C236 is turned ON. Counter C236 has an assigned input of X1, again the input X20 is NOT the counted input.
The assignment of counters and input devices is dependent upon the PLC selected. This is explained in the relevant, later sections.

Driving high speed counter coils:

- The counted inputs are NOT used to drive the high speed counter coils. This is because the counter coils need to be continuously driven ON to reserve the associated high speed inputs.
  Therefore, a normal non-high speed drive contact should be used to drive the high speed counter coil.
  Ideally the special auxiliary contact M8000 should be used. However, this is not compulsory.



4.11.2 Availability of High Speed Counters

FX1S	FX1N	FX2N	FX2NC

The following device table outlines the range of available high speed counters.

INPUT	1 Phase counter user start/reset						1 Phase counter assigned start/reset					2 Phase counter bi-directional					A/B Phase counter				
	C235	C236	C237	C238	C239	C240	C241	C242	C243	C244	C245	C246	C247	C248	C249	C250	C251	C252	C253	C254	C255
X0	U/D						U/D			U/D		U	U		U		A	A		A	
X1		U/D					R			R		D	D		D		B	B		B	
X2			U/D					U/D			U/D		R		R			R		R	
X3				U/D				R		S	R			U		U			A		A
X4					U/D				U/D					D		D			B		B
X5						U/D			R					R		R			R		R
X6										S					S					S	
X7											S					S					S

Key:
U - up counter input **D** - down counter input
R - reset counter (input) **S** - start counter (input)
A - A phase counter input **B** - B phase counter input
C235 - Counter is backed up/latched

Input assignment:

- X6 and X7 are also high speed inputs, but function only as start signals. They cannot be used as the counted inputs for high speed counters.
- Different types of counters can be used at the same time but their inputs must not coincide. For example, if counter C247 is used, then the following counters and instructions cannot be used;
  C235, C236, C237, C241, C242, C244, C245, C246, C249, C251, C252, C254, I0❑❑, I1❑❑, I2❑❑.

Counter Speeds:

- General counting frequencies:
  - Single phase and bi-directional counters; up to 10 kHz.
  - A/B phase counters; up to 5 kHz.
  - Maximum total counting frequency (A/B phase counter count twice)
    FX1S & FX1N 60kHz, FX2N & FX2NC 20kHZ.

- For FX2N & FX2NC Inputs X0 and X1 are equipped with special hardware that allows higher speed counting as follows:
  - Single phase or bi-directional counting (depending on unit) with C235, C236 or C246; up to 60 kHz.
  - Two phase counting with C251; up to 30 kHz.

If any high speed comparison instructions (FNC's 53, 54, 55) are used, X0 and X1 must resort to software counting. In this case, please see the table below:

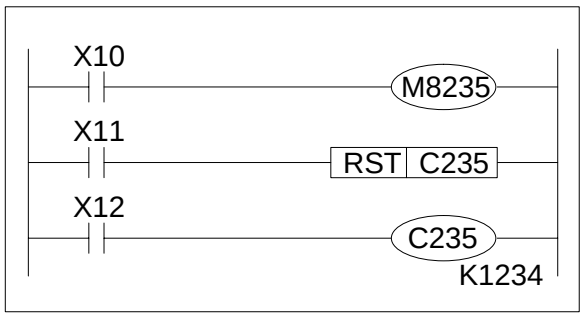
Unit	Function Number	Max. Combined Signal Frequency
FX_{2N} & FX_{2NC}	53 or 54	11 kHz
	55	5.5 kHz
FX_{1S} & FX_{1N}	53 or 54	30 kHz

Calculating the maximum combined counting speed on FX1S:

This is calculated as follows:*(2 phase counter speed x number of counted edges)* ✚ *(the sum of the speeds of the active 1 phase counters).*

4.11.3 1 Phase Counters - User Start and Reset (C235 - C240)

These counters only use one input each. When direction flag M8235 is ON, counter C235 counts down. When it is OFF, C235 counts up.
When X11 is ON, C235 resets to 0 (zero). All contacts of the counter C235 are also reset.
When X12 is ON, C235 is selected. From the previous counter tables, the corresponding counted input for C235 is X0. C235 therefore counts the number of times X0 switches from OFF to ON.



Device specification:

- All of these counters are 32bit up/down ring counters. Their counting and contact operations are the same as normal 32bit up/down counters described on page 4-21. When the counters current value reaches its maximum or setting value, the counters associated contacts are set and held when the counter is counting upwards. However, when the counter is counting downwards the contacts are reset.

Setting range:

- -2,147,483,648 to +2,147,483,647

Direction setting:

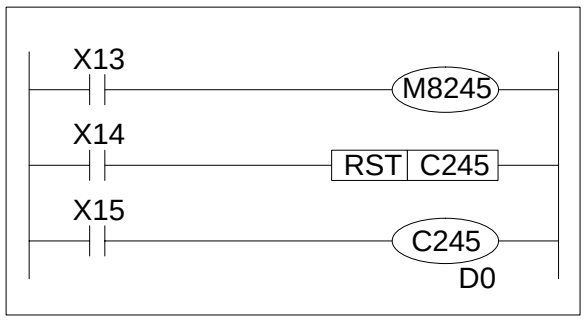
- The counting direction for 1 phase counters is dependent on their corresponding flag M8☆☆☆; where ☆☆☆ is the number of the corresponding counter, (C235 to C240).
  When M8☆☆☆ is ON the counter counts down,
  When M8☆☆☆ is OFF the counter counts up.

Using the SPD instruction:

- Care should be taken when using the SPD applied instruction (FNC 56). This instruction has both high speed counter and interrupt characteristics, therefore input devices X0 through X5 may be used as the source device for the SPD instruction. In common with all high speed processes the selected source device of the SPD instruction must not coincide with any other high speed function which is operating, i.e. high speed counters or interrupts using the same input.
  When the SPD instruction is used it is considered by the system to be a 1 phase high speed counter. This should be taken into account when summing the maximum combined input signal frequencies - see the previous section.

4.11.4 1 Phase Counters - Assigned Start and Reset (C241 to C245)

These counters have one countable input and 1 reset input each. Counters C244 and C245 also have a start input.
When the direction flag M8245 is ON, C245 counts down. When it is OFF C245 will count up.
When X14 is ON, C245 resets in the same manner as normal internal 32bit counters, but C245 can also be reset by input X3. This is assigned automatically when counter C245 is used (see previous counter tables).



Counter C245 also has an external start contact, again automatically assigned. This is actually input X7. Once again this data can be found on the previous counter tables.
When X7 is ON, C245 starts counting, conversely when X7 is OFF C245 stops counting. The input X15 selects and reserves the assigned inputs for the selected counter, i.e. in this case C245.
The reason why these counters use assigned start (X7) and reset (X3) inputs is because they are not affected by the cycle (scan) time of the program. This means their operation is immediate and direct.
In this example C245 actual counts the number of OFF to ON events of input X2.
Note: Because C245 is a 32bit counter, its setting data, specified here by a data register also has to be of a 32bit format. This means that data registers D1 and D0 are used as a pair to provide the 32bit data format required.

Device specification:

- All of these counters are 32bit up/down ring counters. Their counting and contact operations are the same as normal 32bit up/down counters described on page 4-21. When the counters current value reaches its maximum or setting value, the counters associated contacts are set and held when the counter is counting upwards. However, when the counter is counting downwards the contacts are reset.

Setting range:

- -2,147,483,648 to +2,147,483,647

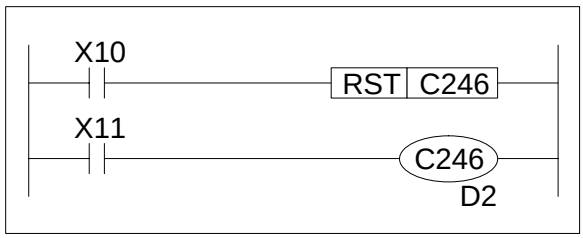
Direction setting:

- The counting direction for 1 phase counters is dependent on their corresponding flag M8☆☆☆; where ☆☆☆ is the number of the corresponding counter, (C241 to C245).
  - When M8☆☆☆ is ON the counter counts down.
  - When M8☆☆☆ is OFF the counter counts up.

4.11.5 2 Phase Bi-directional Counters (C246 to C250)

These counters have one input for counting up and one input for counting down. Certain counters also have reset and start inputs as well.

When X10 is ON, C246 resets in the same way as standard 32bit counters.



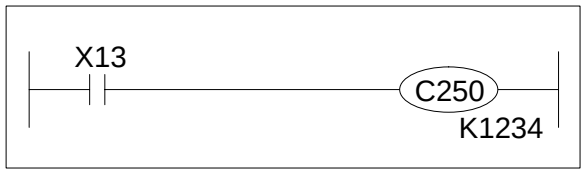
Counter C246 uses inputs;
X0 to count up and
X1 to count down

For any counting to take place the drive input X11 must be ON to set and reserve the assigned inputs for the attached counter, i.e. C246.

Note:

X0 moving from OFF to ON will increment C246 by one
X1 moving from ON to OFF will decrement C246 by one

Bi-directional counter C250 can be seen to have X5 as its reset input and X7 as its start input. Therefore, a reset operation can be made externally without the need for the RST C250 instruction.



X13 must be ON to select C250. But start input X7 must be ON to allow C250 to actually count. If X7 goes OFF counting ceases.
Counter C250 uses input X3 to count up and input X4 to count down.

Device size:

- All of these counters have 32bit operation.

Setting range:

- -2,147,483,648 to +2,147,483,647

Direction setting:

- The counting direction for 1 phase counters is dependent on their corresponding flag M8☆☆☆; where ☆☆☆ is the number of the corresponding counter, (C241 to C245).
  - When M8☆☆☆ is ON the counter counts down,
  - When M8☆☆☆ is OFF the counter counts up.

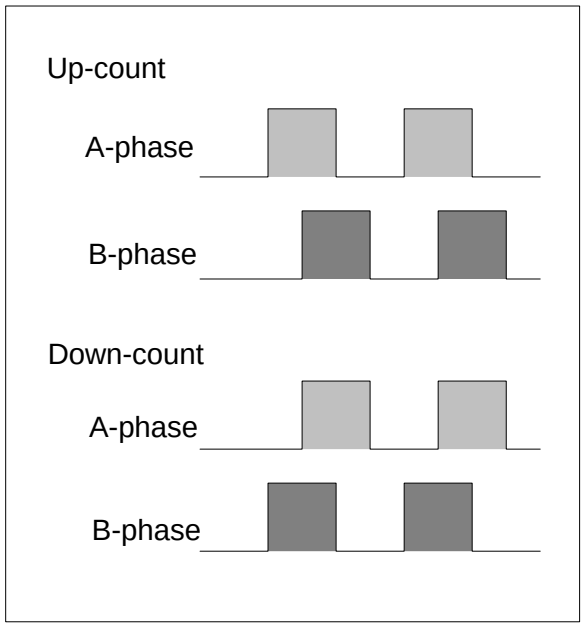
4.11.6 A/B Phase Counters (C252 to C255)

With these counters only the input identified in the previous high speed counter tables can be used for counting. The counting performed by these devices is independent of the program cycle (scan) time. Depending on the counter used, start, reset and other associated inputs are automatically allocated.

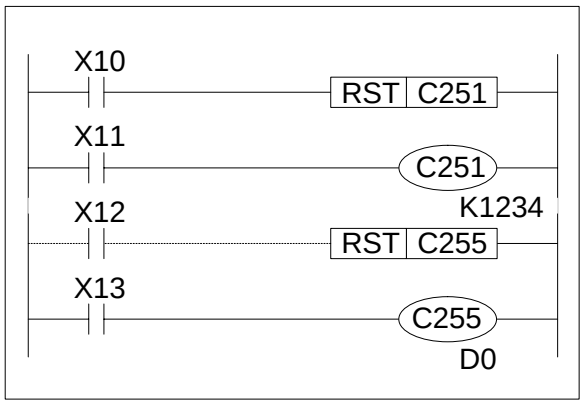
The A phase, B phase input signal not only provide the counted signals but their relationship to each other will also dictate the counted direction.

While the wave form of the A phase is in the ON state and...
the B phase moves from OFF to ON the counter will be counting up.

However, if...
the B phase moves from ON to OFF the counter will be in a down configuration.
One count is registered after both A and B phase inputs have been given and released in the correct order.



C251 counts the ON/OFF events of input X0 (the A phase input) and input X1 (the B phase input) while X11 is ON.
C255 starts counting immediately when X7 is turned ON while X13 is ON. The counting inputs are X3 (A phase) and X4 (B phase).
C255 is reset when X5 is turned ON. It can also be reset with X12 in the sequence.



Device specification:

- A maximum of 2 points - 2 phase, 32bit, up/down counters can be used. The operation of the output contact in relation to the counted data is the same as standard 32bit counters described in section 4.11.

Setting range:

- -2,147,483,648 to +2,147,483,647

Direction setting:

- Check the corresponding special relay M8☆☆☆ to determine if the counter is counting up or down.

4.12 Data Registers

FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: D

Purpose: A storage device capable of storing numeric data or 16/32bit patterns

Alias: Data (register/ device/ word)
D (register)
D
Word

Available forms: General use registers - see page 4-34
Battery backed/latched registers - see page 4-35
Special diagnostic registers - see page 4-35
File registers - see page 4-36
RAM file registers - see page 4-36
Externally adjusted registers - see page 4-37

Devices numbered in: Decimal, i.e. D0 to D9, D10 to D19

Further uses: Can be used in the indirect setting of counters and timers

Example device usage: None

Available devices:

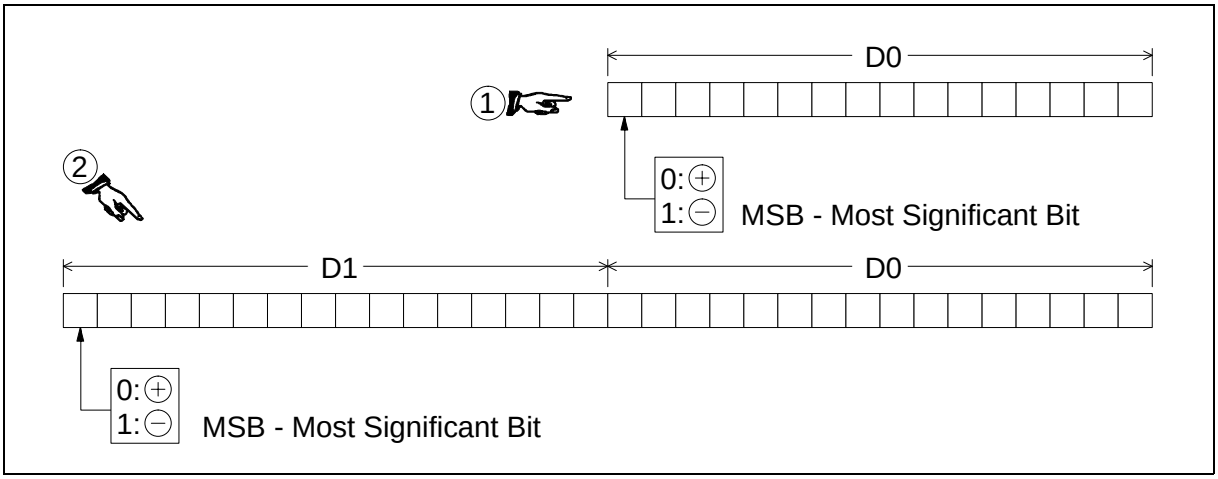
	FX1S	FX1N	FX2N	FX2NC
General use registers	128 (D0 - 127)	128 (D0 - 127)	200 (D0 - 199)	200 (D0 - 199)
Latched registers	128 (D128 - 255)	7872 (D128 - 7999)	7800 (D200 - 7999)	7800 (D200 - 7999)
Diagnostic registers	256 (D8000 - 8255)	256 (D8000 - 8255)	256 (D8000 - 8255)	256 (D8000 - 8255)
File registers R	N/A	7000 (D1000 - 7999)	7000 (D1000 - 7999)	7000 (D1000 - 7999)
Adjustable registers F	2 (D8030 - 8031)	2 (D8030 - 8031)	N/A	N/A

R - These devices are allocated by the user at the expense of available program steps. On FX2N and FX2NC these devices are a subset of the latched registers.

F - These devices are also included under the count for diagnostic registers.

4.12.1 General Use Registers

Data registers, as the name suggests, store data. The stored data can be interpreted as a numerical value or as a series of bits, being either ON or OFF.
A single data register contains 16bits or one word. However, two consecutive data registers can be used to form a 32bit device more commonly known as a double word.
If the contents of the data register is being considered numerically then the Most Significant Bit (MSB) is used to indicate if the data has a positive or negative bias. As bit devices can only be ON or OFF, 1 or 0 the MSB convention used is, 0 is equal to a positive number and 1 is equal to a negative number.



The diagram above shows both single and double register configurations. In the diagram, at point ②, it should be noted that the 'lower' register D0 no longer has a 'Most Significant Bit'. This is because it is now being considered as part of a 32bit-double word. The MSB will always be found in the higher 16 bits, i.e. in this case D1. When specifying a 32 bit data register within a program instruction, the lower device is always used e.g. if the above example was to be written as a 32bit instructional operand it would be identified as D0. The second register, D1, would automatically be associated.
Once the data is written to a general data register, it remains unchanged until it is overwritten. When the PLC is turned from RUN to STOP all of the general data registers have their current contents overwritten with a 0 (zero).

Data retention:

- Data can be retained in the general use registers when the PLC is switched from RUN to STOP if special auxiliary relay M8033 is ON.

Data register updates:

- Writing a new data value to a data register will result in the data register being updated with the new data value at the end of the current program scan.

4.12.2 Battery Backed/ Latched Registers

Once data is written to a battery backed register, it remains unchanged until it is overwritten. When the PLC's status is changed from RUN to STOP, the data in these registers is retained. The range of devices that are battery backed can be changed by adjusting the parameters of the PLC. For details of how to do this please refer to the appropriate programming tools manual.

Using the FX2-40AW/AP:

- When using an FX with either the FX2-40AW or the FX2-40AP a proportion of the latched data registers are automatically assigned for communications use by the FX2-40AW/AP module.

  Communication between Master and Slave 100 points M800 to M899
  10 points D490 to D499

  Communication between Slave and Master 100 points M900 to M999
  10 points D500 to D509

4.12.3 Special Diagnostic Registers

Special registers are used to control or monitor various modes or devices inside the PLC. Data written in these registers are set to the default values when the power supply to the PLC is turned ON.

- Note: When the power is turned ON, all registers are first cleared to 0 (zero) and then the default values are automatically written to the appropriate registers by the system software. For example, the watchdog timer data is written to D8000 by the system software. To change the setting, the user must write the required value over what is currently stored in D8000.

Data stored in the special diagnostic registers will remain unchanged when the PLC is switched from STOP mode into RUN.

Use of diagnostic registers:

- On no account should unidentified devices be used. If a device is used, it should only be for the purpose identified in this manual. Please see chapter 6 for tables containing data and descriptions of the available devices for each PLC.

4.12.4 File Registers

FX1S	FX1N	FX2N	FX2NC

Program memory registers

File registers can be secured in the program memory (EEPROM or EPROM) in units of 500 points. These registers can be accessed with a peripheral device. While the PLC is operating, data in the file registers can be read to the general-use/ battery backed/ latched registers by using the BMOV instruction.
File registers are actually setup in the parameter area of the PLC. For every block of 500 file registers allocated and equivalent block of 500 program steps are lost.

Note: The device range for file registers in the FX1N, FX2N and FX2NC overlaps with the latched data registers. The allocation of these devices as file registers ensures that the data is kept with the program.

Writing to file registers:

- FX1S file register data can only be changed by a peripheral device such as a hand held programmer or a personal computer running the appropriate software.
  For details of how to carry out the changes please reference the relevant operation manual for guidance.
- FX1N, FX2N and FX2NC file register data can also be changed by the program using the BMOV instruction.

Special caution when using FX1S:

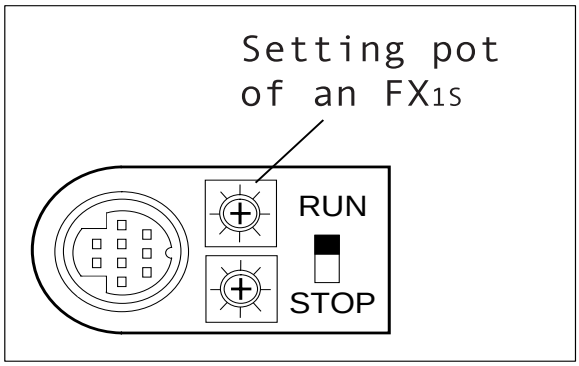
- No file registers can be modified during RUN.

Available devices:

- Please refer to the table on page 4-33 or chapters 6 and 8, where further details of the availability of devices can be found.

4.12.5 Externally Adjusted Registers

The FX1S and FX1N have built in “setting pots” which are used to adjust the contents of certain dedicated data registers. The contents of these registers can range from 0 to 255. This is a built in feature and requires no additional setup or programming.
The FX2N and FX2NC do not have this feature, however, an additional special function unit is available which provides the same function. The unitrequired is the FX2N-8AV-BD. For use, this unit requires the applied instructions VRRD function 85 (Volume Read) and VRSC function 86 (Volume Scale).



	FX1S	FX1N	FX2N	FX2NC
Number of setting pots	2 points, plus 8 points: Supplied by using the additional special function board FX2N-8AV-BD		8 points: Supplied by using the additional special function board FX2N-8AV-BD	
Number of controlled data registers	1: D8030 2: D8031 Additional 8 points selected by the user when applied instructions VRRD and VRSC are used.		Selected by the user when applied instructions VRRD and VRSC are used.	

Uses:

- This facility is often used to vary timer settings, but it can be used in any application where a data register is normally found, i.e. setting counters, supplying raw data, even selection operations could be carried out using this option.

4.13 Index Registers

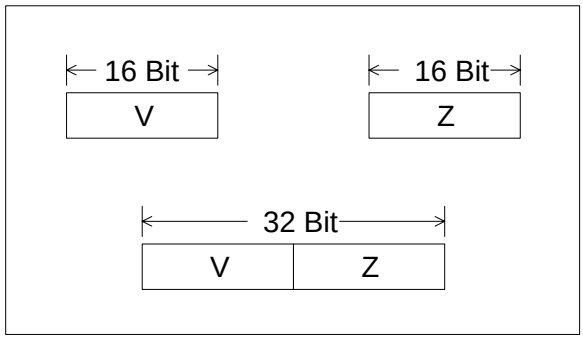
FX1S | FX1N | FX2N | FX2NC

Device Mnemonic: V,Z

Purpose: To modify a specified device by stating an offset.

Alias: (V/ Z) Register
Index (register/ addressing/ modifier)
Offset(s) (register/ addressing/ modifier)
Indices
Modifier

Available forms:
For 16bit data V or Z
(2 devices)
For 32bit data V and Z combined
(1 device - Z is specified)
Operation is similar to data registers.



Devices numbered in: FX1S, FX1N, there are two devices V and/ or Z.
For FX2N and FX2NC there are 16 devices V0 - V7 and Z0 - Z7

Further uses: Can be used to modify the following devices under certain conditions;
X, Y, M, S, P, T, C, D, K, H, KnX, KnY, KnM, KnS

Example device usage:

The program shown right transfers data from D5V to D10Z.
If the data contained in register V is equal to 8 and the data in register Z is equal to 14, then:

V = 8
D5V
D5 +8 =13 ⇨ D13
Z = 14
D10Z
D10 + 14 = 24 ⇨ D24

Hence, the actual devices used after the modifiers V and Z have been taken into account are; D13 and D24 and not D5 and D10 respectively.

Use of Modifiers with Applied Instruction Parameters:

- All applied instruction parameters should be regarded as being able to use index registers to modify the operand except where stated otherwise.

4.13.1 Modifying a Constant

Constants can be modified just as easily as data registers or bit devices. If, for example, the constant K20 was actually written K20V the final result would equal:
K20 + the contents of V

Example:

If V = 3276 then K20V ⇨

K	20
V	(3276)
	3296

4.13.2 Misuse of the Modifiers

Modifying Kn devices when Kn forms part of a device description such as KnY is not possible, i.e. while the following use of modifiers is permitted;

K3Z
K1M10V
Y20Z

Statements of the form:

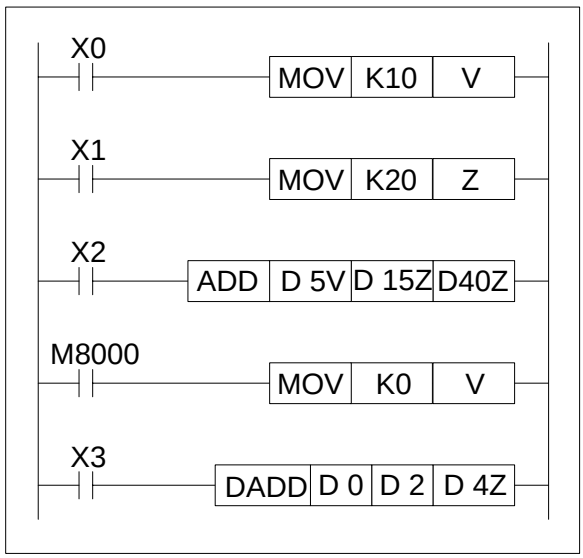
K4ZY30

are not acceptable.

- Modifiers cannot be used for parameters entered into any of the 20 basic instructions, i.e. LD, AND, OR etc.

4.13.3 Using Multiple Index Registers

The use of multiple index registers is sometimes necessary in larger programs or programs which handle large quantities of data. There is no problem from the PLC's point of view in using both V and Z registers many times through out a program. The point to be aware of is that it is sometimes confusing for the user or a maintenance person reading such programs, as it is not always clear what the current value of V or Z is.



Example:

V = 10 (K10)
Z = 20 (K20)
D5V = D15 (D5 + V = D5 + 10 = D15)
D15Z = D35 (D15 + Z = D15 + 20 = D35)
D40Z = D60 (D40 + Z = D40 + 20 = D60)

Both V and Z registers are initially set to K10 and K20 respectively.
The contents of D15 is added to that of D35 and store in D60.
V is then reset to 0 (zero) and both V and Z are used in the double word addition (DADD).
The contents of D1, D0 are then added to D3, D2 and then finally stored in D25, D24.

4.14 Bits, Words, BCD and Hexadecimal

FX1S	FX1N	FX2N	FX2NC

The following section details general topics relating to good device understanding. The section is split into several smaller parts with each covering one topic or small group of topics. Some of the covered topics are;

Available devices:

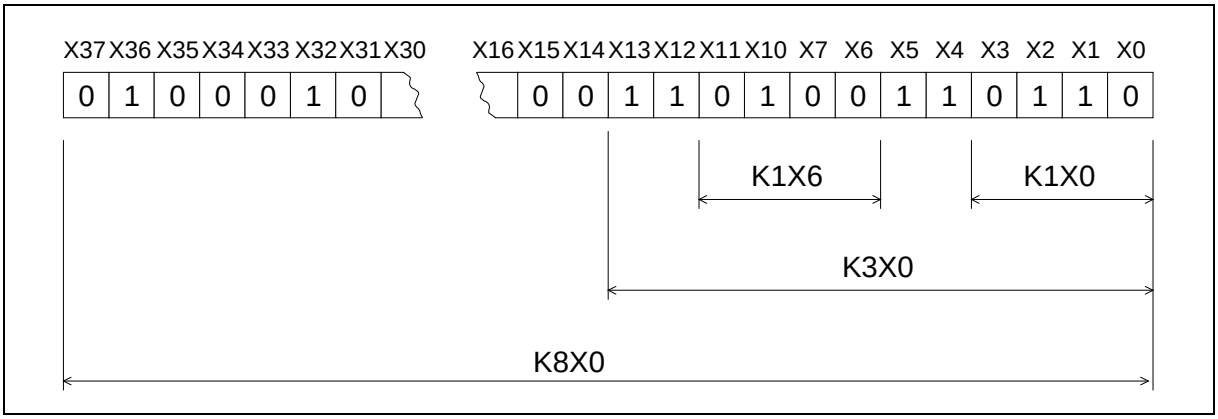
- For PLC specific available devices please see chapter 8.

4.14.1 Bit Devices, Individual and Grouped

Devices such as X, Y, M and S are bit devices. Bit devices are bi-stable, this means there are only two states, ON and OFF or 1 and 0. Bit devices can be grouped together to form bigger representations of data, for example 8 consecutive bit devices are some-times referred to as a byte. Further more, 16 consecutive bit devices are referred to as a word and 32 consecutive bit devices are a double word.
The PLC identifies groups of bit devices which should be regarded as a single entity by looking for a range marker followed by a head address. This is of the form KnP where P represents the head address of the bit devices to be used. The Kn portion of the statement identifies the range of devices enclosed. "n" can be a number from the range 0 to 8. Each "n" digit actual represents 4 bit devices, i.e K1 = 4 bit devices and K8 = 32 bit devices. Hence all groups of bit devices are divisible by 4.
The diagram and example on the following page explain this idea further.....

Assigning grouped bit devices:

As already explained, bit devices can be grouped into 4 bit units. The “n” in KnM0 defines the number of groups of 4 bits to be combined for data operation. K1 to K4 are allowed for 16bit data operations but K1 to K8 are valid for 32bit operations.
K2M0, for example identifies 2 groups of 4 bits; M0 to M3 and M4 to M7, giving a total of 8 bit devices or 1 byte. The diagram below identifies more examples of Kn☆ use.



K1X0 : X0 to X3 ⇨ 4 bit devices with a head address of X0
K1X6 : X6 to X11 ⇨ 4 bit devices with a head address of X6
K3X0 : X0 to X13 ⇨ 12 bit devices with a head address of X0
K8X0 : X0 to X37 ⇨ 32 bit devices with a head address of X0

Moving grouped bit devices:

- If a data move involves taking source data and moving it into a destination which is smaller than the original source, then the overflowing source data is ignored. For example;
  If K3M20 is moved to K1M0 then only M20 to M23 or K1M20 is actually moved. The remaining data K2M24 or M24 to M31 is ignored.

Assigning I/O:

- Any value taken from the available range of devices can be used for the head address ‘marker’ of a bit device group. However, it is recommended to use a 0 (zero) in the lowest digit place of X and Y devices (X0, X10, X20.....etc). For M and S devices, use of a multiple of “8” is the most device efficient. However, because the use of such numbers may lead to confusion in assigning device numbers, it recommended to use a multiple of “10”. This will allow good correlation to X and Y devices.

4.14.2 Word Devices

Word devices such as T, C, D, V and Z can store data about a particular event or action within the PLC. For the most part these devices are 16 bit registers. However, certain variations do have 32 bit capabilities, as can pairs of consecutive data registers or combined V and Z registers.
It may seem strange to quote the size of a word device in bits. This is not so strange when it is considered that the bit is the smallest unit of data within the PLC. So by identifying every thing in bit format a common denomination is being used, hence comparison etc is much easier.
Additional consequences of this bit interpretation is that the actual data can be interpreted differently. The physical pattern of the active bits may be the important feature or perhaps the numerical interpretation of the bit pattern may be the key to the program. It all comes down to how the information is read.

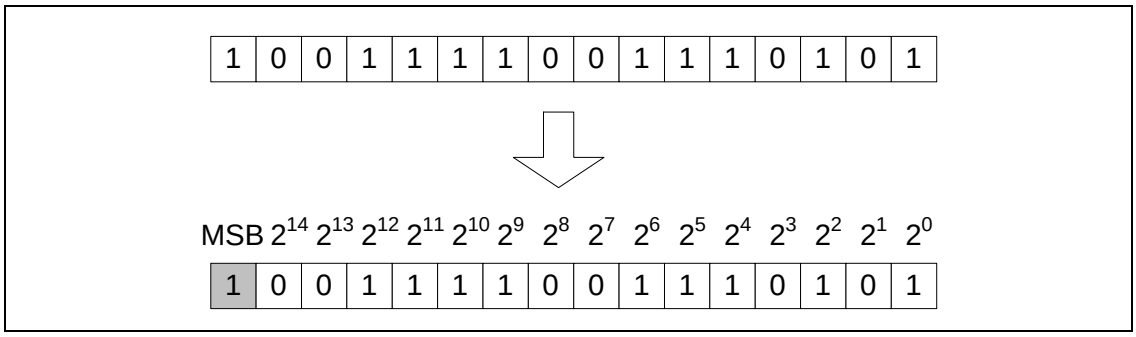
4.14.3 Interpreting Word Data

As word data can be read in many ways the significance of certain parts of the word data can change. PLC's can read the word data as:

- A pure bit pattern
- A decimal number
- A hexadecimal number
- Or as a BCD (Binary Coded Decimal) number

The following examples will show how the same piece of data can become many different things depending wholly on the way the information is read or interpreted.

a) Considering a bit pattern
The following bit pattern means nothing - it is simply 16 devices which have two states. Some of the devices are randomly set to one of the states. However, if the header notation (base 2) is added to the 16 bit data the sum, decimal, total of the active bits can be calculated, e.g.,



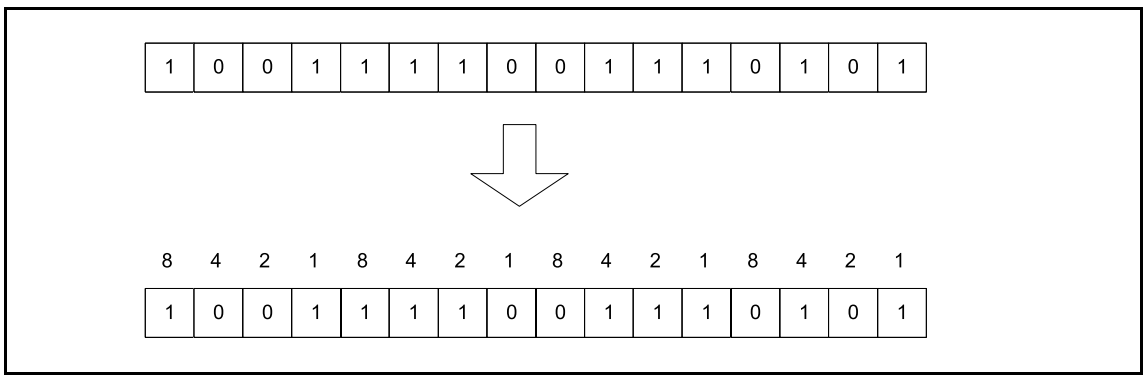
Decimal value = $(2^0 \times 1) + (2^2 \times 1) + (2^4 \times 1) + (2^5 \times 1)$
$+(2^5 \times 1) + (2^9 \times 1) + (2^{10} \times 1) + (211 \times 1) + (2^{12} \times 1)$

Decimal value = 7797

This is in fact incorrect!
There is one bit device which has been shaded in. If its header notation is studied carefully it will be noted that it says MSB. This is the Most Significant Bit. This single bit device will determine if the data will be interpreted as a positive or negative number. In this example the MSB is equal to 1. This means the data is negative.
The answer however, is not -7797.

The reason this is not -7797 is because a negative value is calculated using two's compliment (described later) but can quickly be calculated in the following manner:
Because this is a negative number, a base is set as -32768. This is the smallest number available with 16bit data. To this the positive sum of the active bits is added, i.e. -32768 + 7797.
The correct answer is therefore -24971.
Remember this is now a decimal representation of the original 16 bit - bit pattern. If the original pattern was re-assessed as a hexadecimal number the answer would be different.

b) A hexadecimal view
Taking the same original bit pattern used in point a) and now adding a hexadecimal notation instead of the binary (base 2) notation the bit patterns new meaning becomes:

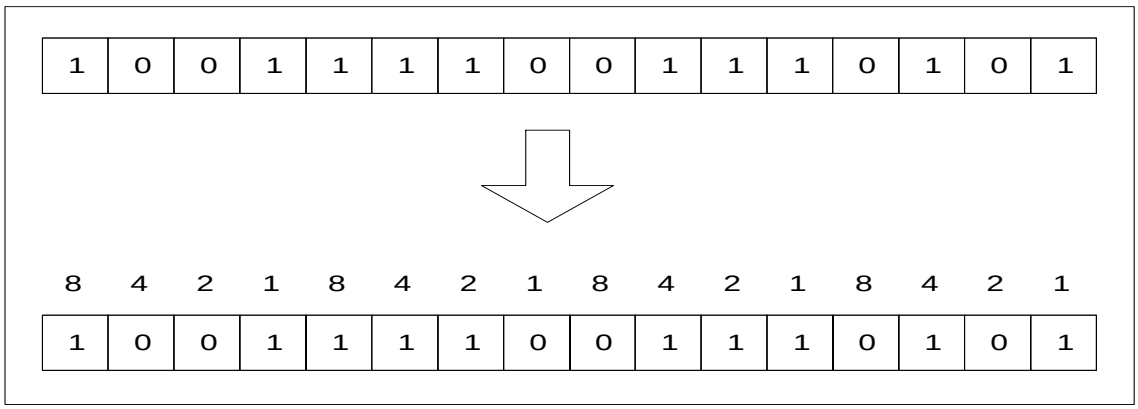


Hexadecimal value = ((1 x 8) + (1 x 1)), ((1 x 8) + (1 x 4) + (1 x 2)),
((1 x 4) + (1 x 2) + (1 x 1)), ((1 x 4) + (1 x 1))

Hexadecimal value = 9E75

Two things become immediately obvious after a hexadecimal conversion. The first is that there is sign bit as hexadecimal numbers are always positive.
The second is there is an "E" appearing in the calculated data. This is actually acceptable as hexadecimal counts from 0 to 15. But as there are only ten digits (0 to 9), substitutes need to be found for the remaining base 16 numbers, i.e. 10, 11, 12, 13, 14 and 15. The first six characters from the alphabet are used as the replacement indices, e.g. A to F respectively.
As a result of base 16 counting, 4 binary bits are required to represent one base 16 or hexadecimal number. Hence, a 16 bit data word will have a 4 digit hexadecimal code.
There is actually a forth interpretation for this bit sequence. This is a BCD or Binary Coded Decimal reading. The following section converts the original bit pattern into a BCD format.

c) ABCD conversion
Using the original bit pattern as a base but adding the following BCD headers allows the conversion of the binary data into a BCD format.



Binary Coded Decimal value= ERROR!!!!!

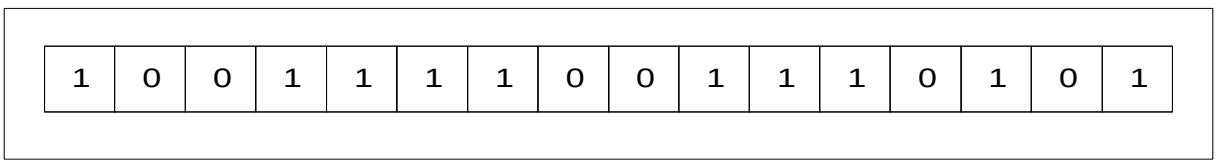
It will be noticed that this will produce an ERROR. The conversion will not be correct. This is because BCD numbers can only have values from 0 to 9, but the second block of 4 bit devices from the left would have a value of 14. Hence, the error.
The conversion process is very similar to that of hexadecimal except for the mentioned limit on values of 0 to 9. If the other blocks were converted just as an example the following values would be found;

Extreme Left Hand Block= ((1 × 8) + (1 × 1)) = 9
Second Right Hand Block= ((1 × 4) + (1 × 2) + (1 × 1)) = 7
Extreme Right Hand Block= ((1 × 4) + (1 × 1)) = 5

BCD data is read from left to right as a normal number would be read. Therefore, in this example the “9” would actually represent “9000”. The second right hand block is actually “70” not “7”. The units are provided by the extreme right hand block, i.e. 5. The hundreds “100’s” would have been provided by the second left hand block (which is in error).
It is also important to note that there is no sign with BCD converted data. The maximum number allowable for a single data word is “9999” and the minimum is “0000”.

Word Data Summary

In each of the previous cases the original bit pattern had a further meaning. To recap the three new readings and the original bit pattern,



Decimal : -24971
Hexadecimal : 9E75
BCD : Error (9?75)

Each meaning is radically different from the next yet they are all different ways of describing the same thing. They are in fact all equal to each other!

4.14.4 Two's Compliment

Programmable controllers, computers etc, use a format called 2's compliment. This is a mathematical procedure which is more suited to the micro processors operational hardware requirements. It is used to represent negative numbers and to perform subtraction operations. The procedure is very simple, in the following example "15 - 7" is going to be solved:

Step1: Find the binary values (this example uses 8 bits)

15	=	00001111
7	=	00000111

Step2: Find the inversion of the value to be subtracted.
Procedure: invert all 1ís to 0ís and all 0ís to 1's.

7	=	00000111
Inverted 7	=	11111000

Step3: Add 1 to the inverted number.
Procedure: add 1 to the right hand most bit. Remember this is binary addition hence, when a value of 2 is obtained 1 is moved in to the next left hand position and the remainder is set to 0 (zero);

Inverted7	11111000
Additional1	00000001
Answer	11111001

This result is actually the same as the negative value for 7 i.e. -7.

Step4: Add the answer to the number the subtraction is being made from (i.e. 15).
Procedure: Remember 1+1 = 0 carry 1 in base 2 (binary).

Original value15	00001111
Answer found in step3	11111001
Solution	(1) 00001000

The "(1)" is a carried "1" and is ignored as this example is only dealing with 8 bits.

Step 5: Convert the answer back.
00001000 = 8

The answer is positive because the MSB (the most left hand bit) is a 0 (zero). If a quick mental check is made of the problem it is indeed found that "15-7 = 8".

In fact no subtraction has taken place. Each of the steps has either converted some data or performed an addition. Yet the answer is correct 15 - 7 is 8. This example calculation was based on 8 bit numbers but it will work equally well on any other quantity of bits.

4.15 Floating Point And Scientific Notation

FX1S | FX1N | FX2N | FX2NC

PLC's can use many different systems and methods to store data.
The most common have already been discussed in previous sections e.g. BCD, Binary, Decimal, Hex. These are what is known as 'integer' formats or 'whole number formats'.
As the titles suggest these formats use only whole numbers with no representation of fractional parts. However, there are two further formats which are becoming increasingly important and they are:

a) Floating point and

b) Scientific notation

Both of these formats are in fact closely related. They both lend themselves to creating very large or very small numbers which can describe both whole and fractional components.

General note:

- Sometimes the words 'Format', 'Mode' and 'Notation' are interchanged when descriptions of these numerical processes are made. However, all of these words are providing the same descriptive value and as such users should be aware of their existence.

Some useful constants

π	3.141×10^{0}
2π	6.283×10^{0}
$\pi/4$	7.853×10^{-1}
π^2	9.869×10^{0}
The speed of light	2.997×10^{8} m/s
Gravity, g	9.807×10^{0} m/s^2
e	2.718×10^{0}

Fixed points:

Boiling point of liquid oxygen	-1.8297×10^{2} °C
Melting point of ice	0.00×10^{0} °C
Triple point of water	1.00×10^{-2} °C
Boiling point of water	1.00×10^{2} °C

4.15.1 Scientific Notation

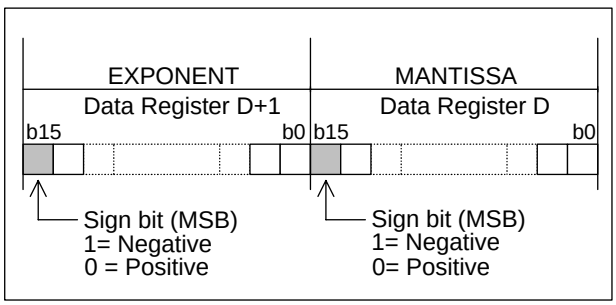
This format could be called the step between the 'integer' formats and the full floating point formats. In basic terms Scientific Notation use two devices to store information about a number or value. One device contains a data string of the actual characters in the number (called the mantissa), while the second device contains information about the number of decimal places used in the number (called the exponent). Hence, Scientific Notation can accommodate values greater/smaller than the normal 32 bit limits, i.e. -2,147,483,648 to 2,147,483,647 where Scientific Notation limits are;

Maximums	Minimums
9999×10^{35}	9999×10^{-41}
-9999×10^{35}	-9999×10^{-41}

Scientific Notation can be obtained by using the BCD, or EBCD in FX2N, instruction (FNC 18 or FNC 118) with the float flag M8023 set ON. In this situation floating point format numbers are converted by the BCD instruction into Scientific Notation - see page 5-22 for details. When using the FX2N the INT instruction (FNC 129) can be used.
Scientific Notation can be converted back to floating point format by using the BIN instruction (FNC 19) with the float flag M8023 set ON - see page 5-22 for details.

The following points should be remembered about the use of Scientific Notation within appropriate FX units;

- The mantissa and exponent are stored in consecutive data registers.
  Each part is made up of 16 bits and can be assigned a positive or negative value indicated by the value of the most significant bit (MSB, or bit 15 of the data register) for each number.



- The mantissa is stored as the first 4 significant figures without any rounding of the number, i.e. a floating point number of value 2.34567×10^3 would be stored as a mantissa of 2345 at data register D and an exponent of 0 (zero) at data register D+1.
- The range of available mantissa values is 0, 1000 to 9999 and -1000 to -9999.
- The range of available exponent values is +35 through to -41.
- Scientific format cannot be used directly in calculations, but it does provide an ideal method of displaying the data on a monitoring interface.

4.15.2 Floating Point Format

Floating point format extends the abilities and ranges provided by Scientific Notation with the ability to represent fractional portions of whole numbers, for example;
Performing and displaying the calculation of 22 divided by 7 would yield the following results:

a) Normal FX operation using decimal (integers) numbers would equal 3 remainder 1

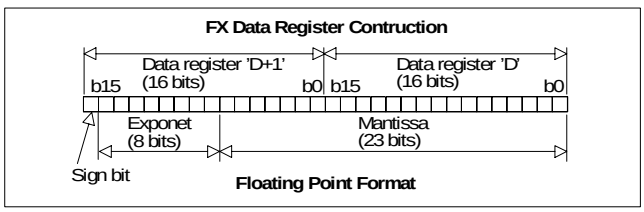
b) In floating point it would equal 3.14285 (approximately)

c) In Scientific format this calculation would be equal to 3142×10^{-3}

So it can be seen that a greater degree of accuracy is provided by floating point numbers, i.e. through the use of larger numerical ranges and the availability of more calculable digits. Hence, calculations using floating point data have some significant advantages. Decimal data can be converted in to floating point by using the FLT, float instruction (FNC 49). When this same instruction is used with the float fag M8023 set ON, floating point numbers can be converted back to decimal. see page 5-49 for more details.
The following points should be remembered about the use of Floating Point within appropriate FX units;

- Floating point numbers, no matter what numerical value, will always occupy two consecutive data registers (or 32 bits).
- Floating point values cannot be directly monitored, as they are stored in a special format recommended by the I.E.E.E (Institute of Electrical and Electronic Engineers) for personal and micro computer applications.
- Floating point numbers have both mantissa and exponents (see Scientific Notation for an explanation of these terms). In the case of floating point exponents, only 8 bits are used.

  Additionally there is a single sign bit for the mantissa. The remaining bits of the 32 bit value, i.e. 23 bits, are used to 'describe' the mantissa value.



Valid ranges for floating point numbers as used in FX Main Processing Units:

Description	Sign	Exponent (bit pattern)	Mantissa (bit pattern)	Remark
Normal Float	0 or 1	11111110 00000001	11111111111111111111111 11111111111111111111110 00000000000000000000001 00000000000000000000000	Largest number +/- 3.403×10^{38} Accuracy: 7 significant figures Smallest number +/- 1.175×10^{-38}
Zero	0 or 1	00000000	00000000000000000000000	All digits are 0 (zero)

4.15.3 Summary Of The Scientific Notation and Floating Point Numbers

The instruction needed to convert between each number format are shown below in a diagrammatically format for quick and easy reference.